2004-03

Session hijacking attacks in wireless local area networks

Onder, Hulusi

Monterey, California. Naval Postgraduate School

NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

SESSION HIJACKING ATTACKS IN WIRELESS LOCAL AREA NETWORKS

by

Hulusi ONDER

March 2004

Thesis Advisor:	Geoffrey XIE
Second Reader:	John GIBSON

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY	**2. REPORT DATE** March 2004	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Session Hijacking attacks in Wireless Local Area Networks			**5. FUNDING NUMBERS**
6. AUTHOR Hulusi ONDER			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT

Wireless Local Area Network (WLAN) technologies are becoming widely used since they provide more flexibility and availability. Unfortunately, it is possible for WLANs to be implemented with security flaws which are not addressed in the original 802.11 specification. IEEE formed a working group (TGi) to provide a complete solution (code named 802.11i standard) to all the security problems of the WLANs. The group proposed using 802.1X as an interim solution to the deficiencies in WLAN authentication and key management. The full 802.11i standard is expected to be finalized by the end of 2004.

Although 802.1X provides a better authentication scheme than the original 802.11 security solution, it is still vulnerable to denial-of-service, session hijacking, and man-in-the-middle attacks. Using an open-source 802.1X test-bed, this thesis evaluates various session hijacking mechanisms through experimentation. The main conclusion is that the risk of session hijacking attack is significantly reduced with the new security standard (802.11i); however, the new standard will not resolve all of the problems. An attempt to launch a session hijacking attack against the new security standard will not succeed, although it will result in a denial-of-service attack against the user.

14. SUBJECT TERMS Wireless Local Area Networks, Authentication, Security, Session Hijacking, 802.1X, 802.11, 802.11i, Encryption, Access Control, Supplicant, Authenticator, Authentication Server, open-source test-bed.			**15. NUMBER OF PAGES** 151
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

SESSION HIJACKING ATTACKS IN WIRELESS LOCAL AREA NETWORKS

Hulusi ONDER
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1998

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2004

Author: Hulusi ONDER

Approved by: Geoffrey Xie, PhD
Thesis Advisor

John Gibson
Second Reader

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Wireless Local Area Network (WLAN) technologies are becoming widely used since they provide more flexibility and availability. Unfortunately, it is possible for WLANs to be implemented with security flaws which are not addressed in the original 802.11 specification. IEEE formed a working group (TGi) to provide a complete solution (code named 802.11i standard) to all the security problems of the WLANs. The group proposed using 802.1X as an interim solution to the deficiencies in WLAN authentication and key management. The full 802.11i standard is expected to be finalized by the end of 2004.

Although 802.1X provides a better authentication scheme than the original 802.11 security solution, it is still vulnerable to denial-of-service, session hijacking, and man-in-the-middle attacks. Using an open-source 802.1X test-bed, this thesis evaluates various session hijacking mechanisms through experimentation. The main conclusion is that the risk of session hijacking attack is significantly reduced with the new security standard (802.11i); however, the new standard will not resolve all of the problems. An attempt to launch a session hijacking attack against the new security standard will not succeed, although it will result in a denial-of-service attack against the user.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my mother, Ayse, and my entire family for their never-ending and full support. I would like to thank my advisor, Professor Geoffrey Xie, for his understanding and guidance throughout my research. I would like to thank my second reader, John Gibson, for his advice and help.

I would also like to express my admiration and thanks to my friend Orhan Ozan, who was my partner for the first part of this thesis, and his spouse, Banu Ozan. Finally, I would like to thank my great friends for their help, support, and friendship.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Wireless networks are an emerging and popular field in the network arena. Since wireless networks reduce the dependence of network clients on established and wire-based infrastructures, they provide mobility and flexibility to the users and have become very increasingly important and pervasive. Wireless networks provide a cable-free internet connection; however, becoming cable-free creates the problems of being open to anybody in the coverage area. Security aspects of wireless networks were examined and implemented after the technology was introduced. The very first design of all the wireless protocols did not consider security issues. Problems were discovered after installation of the systems. These problems were addressed by applying patches and some means of identification or privacy, such as Service Set Identifier (SSID) checks and Wired Equivalence Privacy (WEP). The weaknesses in these first security precautions, however, were exploited shortly after they were introduced.

In addition to these actions to counter the security problems of wireless network traffic, a solid and reliable authentication mechanism should also be used to control the access of the users. The IEEE 802.11i working group was formed to solve the security problems of wireless networks, including the authentication issue. This working group proposed that the 802.1X authentication scheme be used until a complete solution to all the known problems of wireless networks could be developed. The 802.1X authentication mechanism includes the use of an authentication server to control the access of the mobile client trying to use the network.

Since the 802.1X authentication scheme is not especially designed for wireless networks, it specifically does not address the problems of WLANs and carries the problems and security weaknesses of wired networks to the more “open” wireless world. Research [1] conducted in 2002 revealed problems that may be encountered in wireless networks that use the 802.1X authentication scheme. The security vulnerabilities that are present in wireless networks may be used by attackers to conduct several denial-of-service, man-in-the-middle, and session hijacking attacks.

A session hijacking attack, if conducted successfully, can allow the attacker to access the wireless network in the place of a legitimate user by disconnecting that particular user from the network and assuming its identity as far as the network is concerned.

Since the University of Maryland paper [1] was published, there have been two formal replies to the paper: one from Cisco [2] and the other from Orinoco [3]. Both replies have accepted the possibility of session hijacking attack under certain conditions. The session hijacking attack scenario in University of Maryland paper did not mention the encryption that might be used in the wireless networks. The reply papers indicated that if the encryption of the wireless networks were strong enough, then the session hijacking attack would not reach its goals, but would result in a denial of service situation instead.

B. THESIS OBJECTIVES

The main objective of this thesis is to conduct a systematic evaluation of the risks of session hijacking attacks in wireless networks. A major portion of the effort is devoted to an attempt to implement session hijacking attacks over an actual network. As a result, the work provides unique observations from a practical perspective. Where an attack is successfully completed, the security flaws of the system giving rise to the success of that type of attack are examined and possible ways of overcoming those flaws are described in the thesis. Otherwise, the protective mechanism responsible for preventing the attack is identified.

An open-source 802.1X wireless networking test-bed is required in this thesis. One such test-bed was first implemented by LTJG H. Selcuk Ozturk, Turkish Navy, in 2002 [4]. As a part of this thesis, the same test-bed is built again with the latest versions of the software components.

C. THESIS ORGANIZATION

This thesis is organized into six chapters. Chapter II covers the issues related to the 802.1X authentication standard. The entities and the protocols of the standard are

broadly discussed. Chapter III explains the structure of the test-bed and installation of the software to build the test-bed. Chapter IV contains the experiments and results of the experiments conducted on the test-bed pertinent to the session hijacking attack. Chapter V introduces the new security standard (802.11i). The final part of Chapter V discusses the new standard in regard to the weakness that result in a session hijacking attack. Finally, Chapter VI contains the conclusions and the areas for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND AND SESSION HIJACKING ATTACKS

A. INTRODUCTION

The growing uses of wireless networks in everyday life generate urgent needs for confidentiality, privacy and access control. Authentication and Access Control issues are insufficiently covered and addressed in the original 802.11 protocol. After the discovery of the 802.11 protocol problems, the IEEE 802.1X standard became a temporary solution until a precise and complete solution is developed by the IEEE 802.11i working group, which has been established particularly to address the security problems of wireless networks.

802.1X is known as the standard for Port-Based Network Access Control. This standard is intended to provide sufficient solutions to authentication, access control and key management issues. However, a University of Maryland paper [1], published in 2002, asserts that the 802.1X standard has shortcomings, which could be exploited by attackers to launch successful man-in-the-middle, session hijacking and denial of service attacks. After the paper was published, Cisco and Orinoco presented two formal responses [2, 3]. These companies are among the top wireless networks equipment producers. Interestingly, the responses did not totally dismiss the possibility of successful man-in-the-middle and session hijacking attacks as asserted by the University of Maryland paper [1].

The 802.11i working group (TGi) approved the seventh draft of the new 802.11i standard in November 2003. The standard addresses the problems of wireless networks. The weakness in WEP encryption and the lack of mutual authentication issues will be addressed when the standard is finalized. 802.1X will remain the authentication mechanism of the new standard.

This chapter covers the 802.1X authentication mechanism in detail, including the protocols used for communication between the participating entities. The message sequence of the authentication is examined in detail. This chapter also discusses the session hijacking attack. The aspects of the University of Maryland paper [1] and the responses [2, 3] related to the session hijacking attack are also examined.

B. IEEE 802.1X AUTHENTICATION MECHANISM

1. Introduction

Since the wireless environment is not as restricted to outside users as are wired networks, a trusted security framework should be established to control the access of the users and authenticate the pre-approved legitimate users. The 802.1X standard [12] is the authentication and access control standard, which was approved by the IEEE in 2001. The 802.1X standard was not intended to be used by only wireless networks. It was meant to be used by all 802 standard networks, such as contention-based bus networks (802.3), FDDI and Token Ring (802.5). After the vulnerabilities of the wireless networks were unveiled, the 802.1X standard was proposed to be used in wireless networks until the 802.11i working group (TGi) finds a complete solution.

2. Elements of the Authentication Mechanism

The security framework of the 802.1X standard [12] consists of three main entities: the supplicant, the authenticator, and the authentication server. Figure 1 shows the entities of 802.1X

a. Supplicant

The supplicant is an entity that desires to use the services offered by the authenticator. Thus the client side of the wireless network is labeled the "supplicant." It is an entity at the one end of the network that is authenticated by the authentication server on the other end of the network.

b. Authenticator

The authenticator is the entity at one end of a point-to-point LAN segment that facilitates authentication of another entity attached to the other end of that link.[12] The authenticator has two roles: one before the authentication and the other after the authentication. The authenticator relays the authentication packets between the supplicant and the authentication server. After a successful authentication takes place, the authenticator provides network connectivity to the supplicant independent of the authentication server.

c. Authentication Server

The authentication server is an entity that provides authentication service to an authenticator. This service determines from the credentials provided by the supplicant, whether the supplicant is authorized to access the network services provided by the authenticator [12]. An authentication server is the authority in a network that decides the access of the supplicants according to their credentials. Authentication is the only function for this server. After a successful authentication, the server is dormant until another supplicant wants to use the network.

3. Application of the 802.1X Standard for WLANS

The authenticator of the wireless network plays a key role in the access management of the network. The authenticator is the middle entity that controls the gates of the network by its ports, which can be considered as the logical connection between the authenticator and supplicant.

There are two different ports defined in the 802.1X standard: a controlled and an uncontrolled port. The uncontrolled port is used for the authentication and the controlled port is used for network connections to the authenticated supplicants.

At the very beginning of the authentication process, the authenticator relays the management frames between the supplicant and the authentication server. The first communication of any supplicant is over the uncontrolled port for the authentication. The controlled port is kept closed until a successful authentication occurs. Once the supplicant is verified and access to the network is granted, the controlled port is opened to the supplicant. The supplicant can reach and use the network services only via the controlled port.

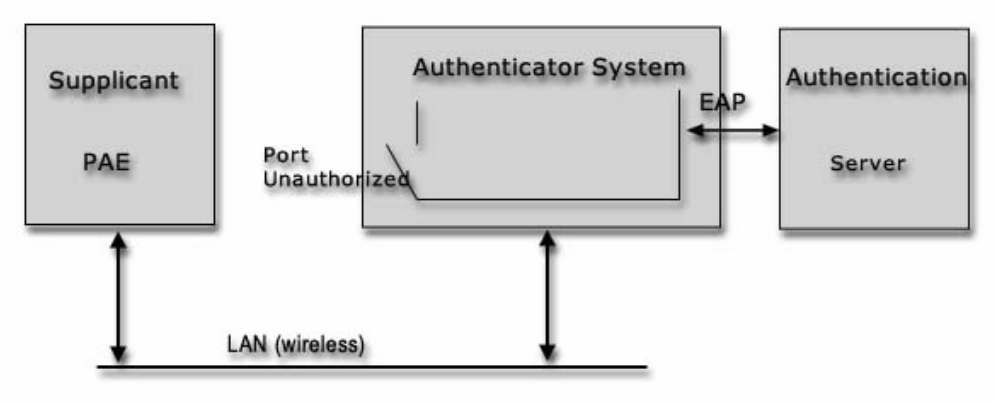


Figure 1. Authenticator, Supplicant, and Authentication Server Roles (From Ref. 4)

The 802.1X authentication standard is intended to be used in Local Area Networks. The terms used in the standard documentation does not specifically mention the Wireless Local Area Networks. Until a complete solution to the problems of the wireless networks is completed by the 802.11i working group (TGi), 802.1X is proposed to be used for authentication in WLANs. The entities in a WLAN and in a traditional LAN can be mapped for the implementation of the 802.1X implementation. The communication protocols between the entities will remain the same, and they will be mentioned later in this chapter.

In a WLAN environment, each mobile network client is a supplicant, the access point serves the authenticator role, and the authentication server role is assigned to a Remote Authentication Dial-in Server (RADIUS). In a larger application, the authentication server's role may be divided into more entities. Additionally, multiple authenticators may be used to provide services to more supplicants.

4. Protocols Used by the 802.1X Standard

Three protocols are mentioned in the 802.1X standard. The most important protocol is the Extensible Authentication Protocol (EAP) which is used for the authentication between two entities. EAP over LAN (EAPOL) is another protocol defined in the 802.1X protocol. It is basically used to carry the EAP packets between the supplicant and the authenticator. In other words, the EAPOL protocol is the encapsulation technique of the EAP messages in LAN environments between the authenticator and the supplicant. The final protocol that is used in this authentication method is the RADIUS

protocol which carries the EAP packets between the authenticator and the authentication server. The protocol stack is pictured in Figure 2.

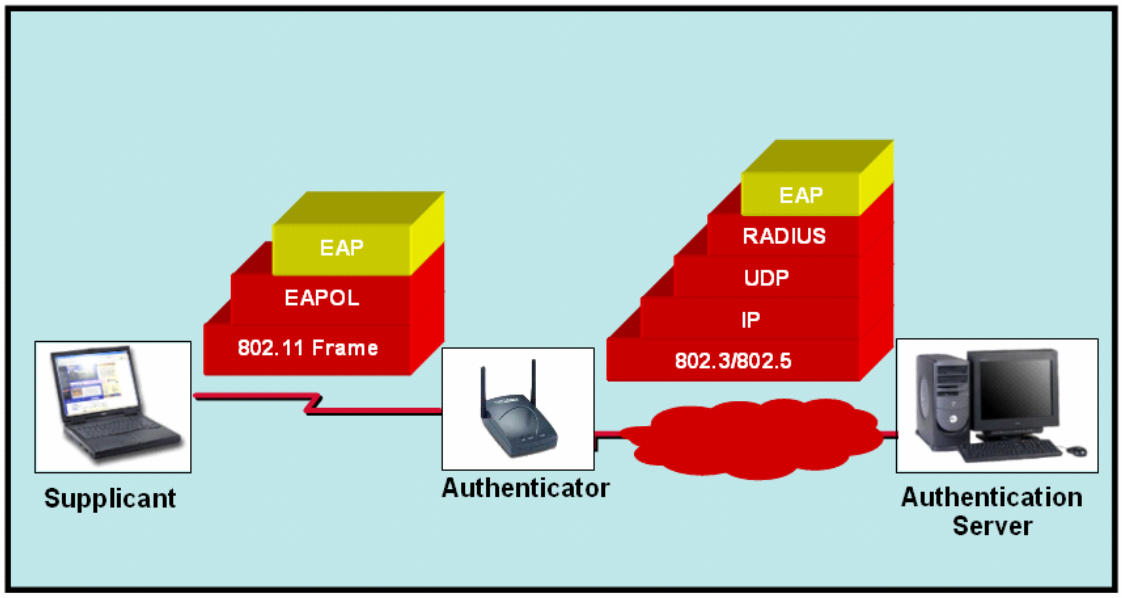


Figure 2. The 802.1X Client Authentication Protocol Stack in the 802.11 Framework (From Ref 4)

The details of these three important protocols will be described in this chapter. The details of the protocols will explain the vulnerabilities of the entire authentication method.

a. Extensible Authentication Protocol over LAN (EAPOL)

The EAPOL protocol is the encapsulation technique that is used to carry the EAP packets between the supplicant and the authenticator. In the IEEE protocol standard [7], EAPOL is described for Ethernet (802.3) and Token Ring/FDDI MAC addresses. However, EAPOL encapsulation used with Ethernet MAC can be applied to other LAN technologies that share the same basic format as Ethernet. It is convenient to use this protocol for Wireless LAN applications. The packet format of the EAPOL is depicted in Figure 3.

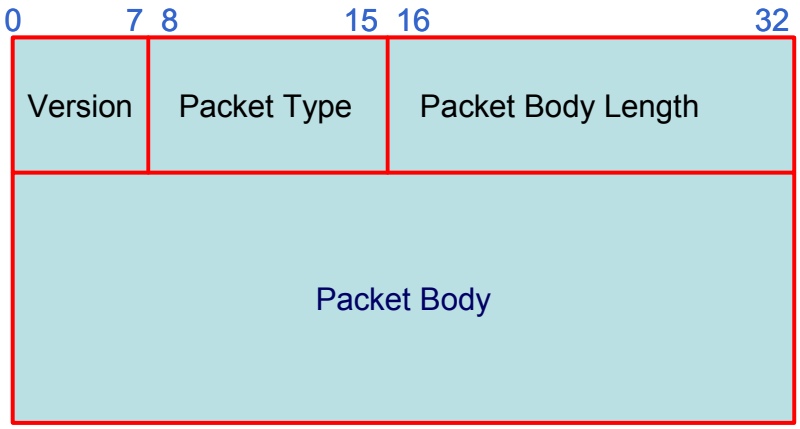


Figure 3. EAPOL Protocol Format (From Ref. 12)

The following briefly describes the fields of the EAPOL protocol:

(1) Version: The value of this field identifies the supported version of the EAPOL protocol supported by the sender. EAPOL version 1.0 is the current version. Since this field is 8 bits long, the version is represented as 0000 0001.

(2) Packet Type: This field identifies the type of packet that the sender is transmitting. There are five possible packets that can be sent. Four of these packets are about the type of the EAPOL protocol and the last one is about the payload of the packet, which is an EAP packet. The values of the field according to the packet type are listed in Table 1. All the other possible values of this field are not used and are reserved for possible future extensions of the protocol.

Packet Type	Definition	Value
EAP-Packet	The frame carries an EAP packet	0000 0000
EAPOL-Start	The frame is an EAP-Start packet	0000 0001
EAPOL-Logoff	The frame is an explicit EAPOL-logoff request frame	0000 0010
EAPOL-Key	The frame is an EAPOL-Key frame	0000 0011
EAPOL-Encapsulated-ASF-Alert	The frame carries an EAPOL-Encapsulated-ASF-Alert	0000 0100

Table 1. Packet Types of EAPOL Protocol (From Ref. 12)

(3) Packet Length: The total length of the packet is represented in octets in this field.

(4) Packet Body: The packet body contains data if the packet type is other than the EAP-Start and EAP-Logoff. These two packets do not contain any data portion. The packet body contains the EAP packet if the packet type is an EAP-Packet. The field contains the EAP-Key Descriptor if the packet type is an EAPOL-Key. Finally, if the packet type is EAPOL-Encapsulated-ASF-Alert, the Packet Body contains the ASF alert frame.

For any entity in the network to process an EAPOL packet, the destination MAC address of the packet should contain the address of the receiving entity, the LAN type should match the MAN type of the receiver, and the packet field should contain the values specified in Table 1. After all these three criteria have been matched, the packet can be evaluated as a regular EAPOL packet.

b. Extensible Authentication Protocol (EAP)

The Point-to-Point Extensible Authentication Protocol (EAP) is a mechanism that provides a standard message exchange mechanism for devices using an agreed upon authentication protocol. EAP protocol uses the link layer for communication. Since it does not require the devices to have IP addresses for communication, the EAP is used as a base technology for both wired and wireless networks for authentication. After a successful authentication, the devices are assigned a legitimate IP number by the DHCP server of the network.

EAP was first developed for use with PPP in RFC 2284 and since then it has been widely deployed. EAP serves as a base technology and protocol for authentication mechanisms. EAP does not provide authentication all by itself. EAP supports a variety of authentication protocols to provide security during the authentication process. Some of the authentication protocols that are used with EAP are EAP-MD5, EAP-TLS, EAP-TTLS, EAP-PEAP and CISCO-Leap.

EAP-TLS is a widely used certificate-based authentication protocol. It is the authentication protocol that will be used by the test-bed built for this thesis. Standard documentation of this protocol can be found in RFC-2716 [23]. EAP-TLS provides strong security between the supplicant and the authentication server through the use of

PKI certificates. While providing a very secure way of authentication, the complexity and the overhead of using PKI certificates are the main drawbacks of this authentication method.

The EAP protocol is adopted by the IEEE 802.1x standard to provide an authentication mechanism for the 802.11 standard. The packet format of the EAP protocol is shown in Figure 4.

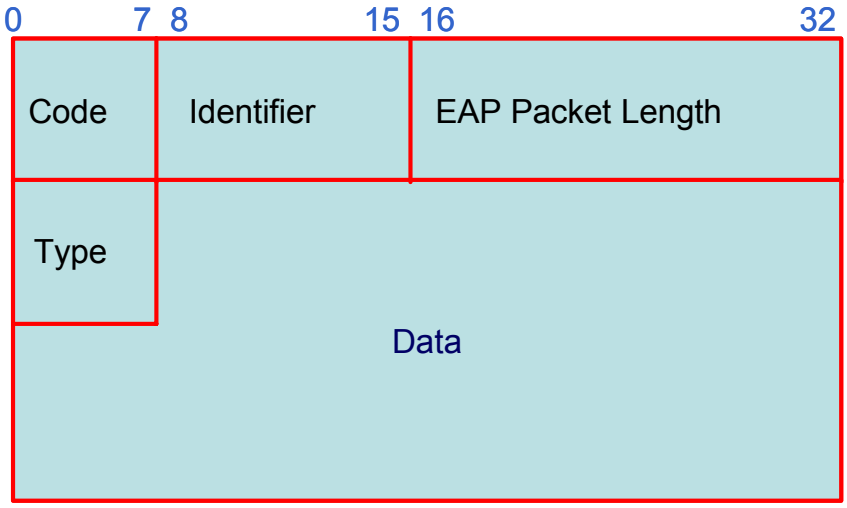


Figure 4. EAP Packet Format (From Ref. 23)

(1) Code: This field identifies the type of the EAP packet, which could be one of the following EAP packets; 1-Request, 2-Response, 3-Success, 4-Failure.

(2) Identifier: This field is one octet long and allows a matching of responses with requests. This field, with the system port, uniquely identifies an authentication exchange. The value of this field is determined by the operation of the authenticator device used in a new EAP-Request/Identity frame. This identity is used by the supplicant and the authenticator throughout the authentication process.

(3) Length: This field is two octets long and the value indicates the length of the whole EAP packet including the code, identifier, length and data fields.

(4) Type: This field indicates the authentication protocol. Several authentication protocols are supported by EAP: Transport Layer Security (TLS), MD5, One Time Passwords (OTP), and Light-weight EAP (LEAP).

(5) Data: The data field may contain zero or more octets depending on the type of the EAP packet indicated in the Code filed.

c. Remote Authentication Dial in User Service (RADIUS)

The RADIUS protocol facilitates centralized user administration, authentication, authorization, and accounting for network access. It was originally developed for dial-up remote access but became widely popular among other network access types, including wireless networks. RADIUS is used to carry the EAP packets between the authenticator and the authentication server. RADIUS provides per packet authenticity and integrity by using a shared secret and a calculation algorithm. The shared secret is an alphanumeric value used to calculate the MD5 sums for each packet.

EAP packets are encapsulated inside the RADIUS packets. The authenticator is not required to know the type of authentication since the EAP packets travel between the supplicant and the authentication server. The RADIUS client (authenticator) sends the user's credentials and the connection parameters to the RADIUS Server. The RADIUS server checks the incoming RADIUS packet and returns the response in a RADIUS packet. The EAP packet, which resides inside the RADIUS packet, is extracted and sent to the supplicant. RADIUS packets are sent via UDP. UDP Port 1812 is used for RADIUS authentication messages and UDP port 1813 is used for RADIUS accounting messages. The packet format of the RADIUS packet is defined in Figure 5.

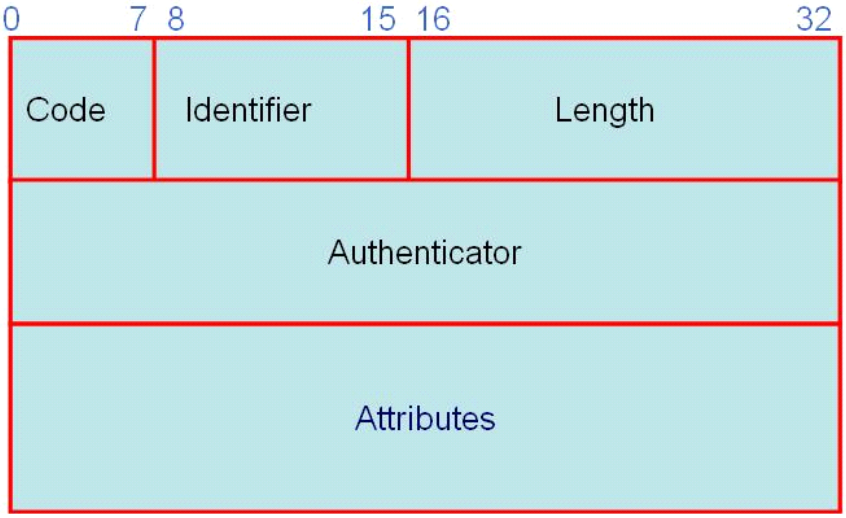


Figure 5. RADIUS Packet Format (From Ref. 24)

(1) Code: This field identifies the type of the RADIUS packet. Table 2 includes the complete list of the codes with their respective descriptions and the sender of the message. These codes are defined in RFCs 2865 [24] and 2866 [25].

Value	Description	Sender
1	Access-Request	RADIUS Client
2	Access-Accept	RADIUS Server
3	Access-Reject	RADIUS Server
4	Accounting-Request	RADIUS Client
5	Accounting-Response	RADIUS Server
11	Access-Challenge	RADIUS Server
12	Status-Server (experimental)	Reserved
13	Status-Client (experimental)	Reserved

Table 2. Code Field Values and Descriptions of a RADIUS Packet (From Ref. 24)

The RADIUS Server sends the Access-Accept message if the connection attempt is authenticated and authorized. The RADIUS server sends an Access-Reject message if the credentials are not authentic or if the connection attempt is not authorized.

(2) Identifier: This field is one octet long and allows the RADIUS client to match responses with requests. If the RADIUS server receives two request messages from the same IP address, the source UDP port and the same identifier in a short span of time, this packet is evaluated as a duplicate.

(3) Length: This field shows the complete length of the RADIUS packet including all the fields.

(4) Authenticator: This value is used to authenticate the reply from the RADIUS server and is used in the password hiding algorithm. The request authenticator is a 16-octet long random number used in the Access-Request packets. This value passes through the MD5 hash algorithm and the XOR operation with the shared secret and other field values of the Response packet and returns to the client in the form

of a response authenticator. This value enforces the per-packet authenticity and integrity verification.

(5) Attributes: The attributes field carries the specific authentication, authorization information and the configuration details for the request and reply. Since the RADIUS Protocol supports a variety of authentication mechanisms, the content of this field varies. For the EAP authentication mechanism, this field contains EAP attributes. The most important attributes are the user name, user-password, NAS server IP address and port number, and the service type.

5. 802.1X Authentication Procedure

A complete authentication session uses all the protocols defined above. The authentication can be completed by the successful transfer of all the packets among the three main entities. Figure 6 illustrates the authentication. Network monitoring tools like Ethereal is employed in the implementation phase of the test-bed for troubleshooting and debugging purposes. The details of the authentication message sequence are as follows:

1) Authentication sequence is started by the supplicant with an EAPOL-Start packet sent to the authenticator.

2) The authenticator responds with an EAP-Request/Identity packet. The EAP traffic between the supplicant and the authenticator is encapsulated in EAPOL packets.

3) The supplicant replies to the EAP-Request/Identity packet with an EAP-Response/Identity packet. On receiving this packet, the authenticator extracts the EAP packet from the EAPOL packet and places it into a RADIUS Access-Request packet and sends it to the authentication server.

4) The authentication server keeps track of a database of the legitimate authenticators. The authentication server checks the user ID of the authenticator and verifies it.

5) The authentication server sends back a RADIUS-Access Challenge message if the user ID of the supplicant is in the database of the authentication server.

6) The authenticator relays the EAP-Request message embedded in the access challenge message that came from the authentication server to the supplicant in an EAPOL-EAP packet. The EAP-Request/Response messages are sent and received until the authentication is completed. (7, 8, 9, 10)

11) Finally a success or failure message will be sent by the authentication server to the supplicant. This packet will be relayed by the authenticator to the supplicant in the form of an EAP-Success/Failure packet.

12) After a successful authentication, the authenticator opens up the controlled port to the supplicant. In the case of a failure in authentication, the supplicant is disassociated from the uncontrolled port.

13) After a successful authentication, a key can be distributed by an EAPOL-Key message to enforce privacy.

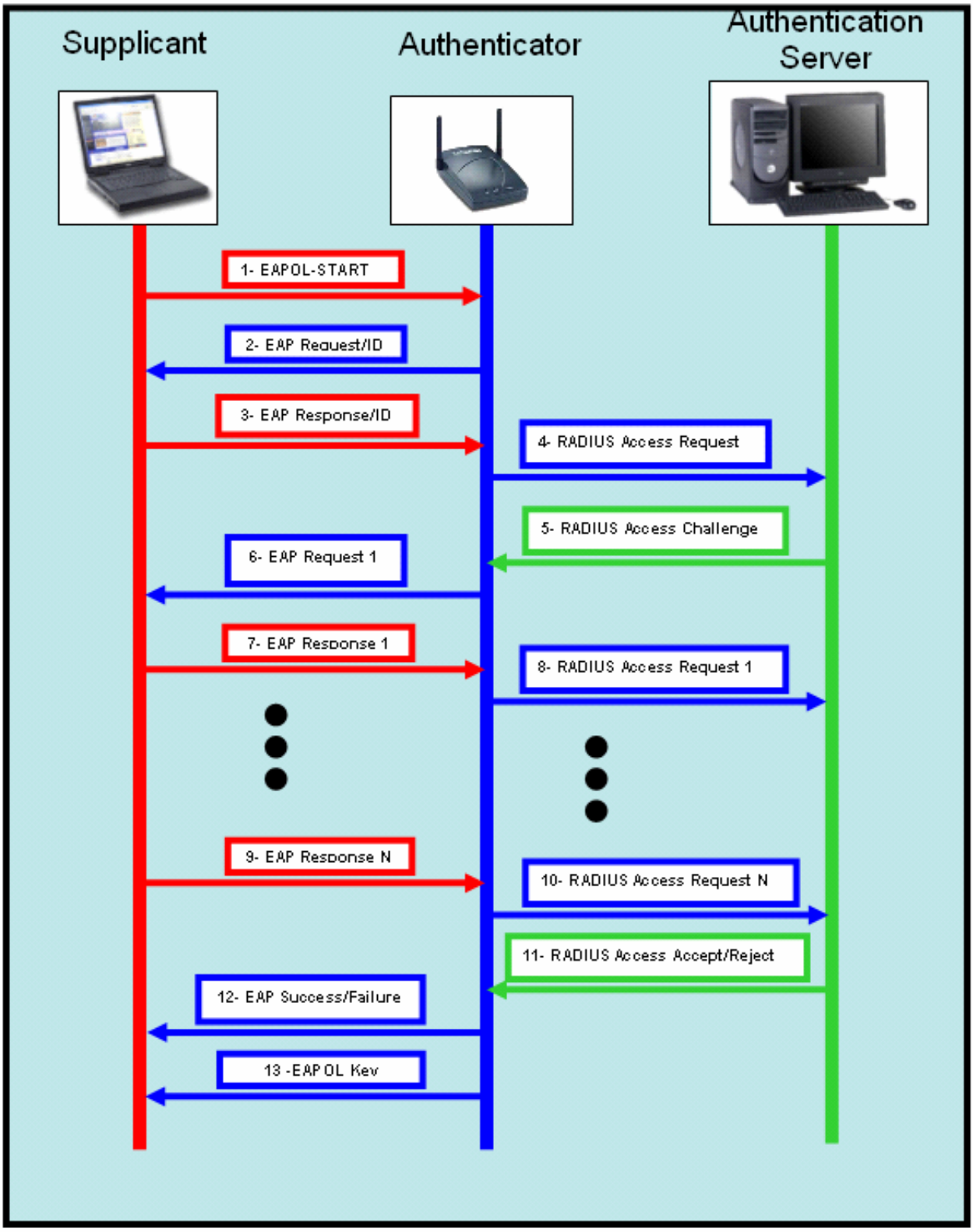


Figure 6. The 802.1X Authentication Session (From Ref. 4)

C. SESSION HIJACKING ATTACK

1. Introduction

It is helpful to explain the session hijacking attack concisely before going into the details of the standards and the actual attack scenario.

The ultimate goal of the session hijacking attack is usurping a legitimate user and obtaining the privileges of that particular user to gain access to the network. Once a successful session is established between a legitimate user and the network, the rogue user captures the traffic between the two entities. After extracting the necessary information from the traffic, the attacker explores holes in the communication protocol to launch the attack. After obtaining enough information about the connection between the network and the user, the attack takes the place of the legitimate user by disconnecting the user.

The session hijacking attack can be divided into two main parts. The first part of the attack is the disassociation of the regular user. The attacker can achieve this by posing as the authenticator and creating and sending fake logoff messages to the user. The second part of the attack is using the privileges of the user, which are obtained in the previous steps of monitoring, to access the network. Since the authentication server is out of the scene after the authentication takes place at the beginning of the session, mimicking the legitimate user is easy for the attacker. The authentication mechanism and its challenges are overcome at the end of the attack. The legitimate user is disassociated from the network and still unaware of the particular attack. The attacker obtains access to the network until the end of the session. The authenticator is also unaware of the attacker because the attacker uses the same credentials, such as the MAC address, as the legitimate and previously authenticated client.

2. Vulnerabilities of the 802.1X Authentication Scheme

The vulnerabilities of the 802.1X authentication mechanism will be covered from the session hijacking point of view. The university of Maryland paper [1] describes the vulnerabilities that can cause a possible session hijacking attack. The following examines the main vulnerabilities:

a. Propagation Medium

Since the medium that carries the packets of the wireless network is the air, limiting the RF signal availability within a specific region is difficult. Eavesdropping is the starting point of the attacks against wireless networks. To overcome this vulnerability, which is a result of the broadcast nature of wireless networks, the authentication to the network and the confidentiality of the traffic should be reliable.

b. Miscommunication of the State Machines

The loose consistency between the 802.1X and 802.11 state machines provides the means for a session hijacking attack in wireless networks.[1] Since no clear communication between these two state machines exists, an attacker may change one of the state machines to launch the attack.

Upon the supplicant's successful authentication, which occurs after the access point sends the EAP-Success frame to the supplicant, the supplicant's state machine transitions to the authenticated state. The authenticated state is entered from any other states of the supplicant on receipt of an EAP-Success frame. This state is maintained until the client disconnects from the network or an EAP-Request/Identity frame is received from the authenticator. The detailed picture of the supplicant state machine is shown in Appendix F. The attacker could forge the MAC address of the authenticator and send a MAC disassociate frame to the supplicant and thereby change the supplicant's state to unassociated. The supplicant cannot reach the network until it is associated again. [12]

The supplicant transitions to the acquired state where the supplicant waits for an EAP-Request frame from the authenticator to initiate the authentication again. By transitioning to this state, the state machine sets a timer called "authWhile" and sends a response identity frame. This timer lets the supplicant wait for a particular amount of time for the EAP-request frame from the authenticator. (The default wait time is 60 seconds). When the timer expires, the supplicant returns to the very beginning of the authentication procedure and starts all over again. [12]

There is another state machine for the same supplicant, which the authenticator maintains. This is called the Authenticator PAE state machine. After successful authentication between the supplicant and the authenticator, the Authenticator PAE state machine transitions to the authenticated state. There are only two types of frames that can change the authenticated state of this state machine. One of them is an EAPOL-logoff frame sent by the supplicant. On receiving this frame, the authenticator PAE state machine transitions to the disconnected state and closes the controlled port. The other one is the reauthentication request or an EAPOL-start frame sent by the supplicant. Upon receiving this frame, the authenticator PAE state machine transitions to a connecting state in order to reauthenticate. [12]

After analyzing both state machines, the lack of communications between them can be seen easily.(The diagrams of the state machines are presented in Appendix F) Once the attacker changes the supplicant state machine, the authenticator PAE state machine is virtually unaware of the recent change. The authenticator PAE's state machine remains in an authenticated state until the supplicant tries to authenticate again. [12]

The attacker can keep the legitimate supplicant in the same acquired state by sending EAP-Request/Identity frames. On receiving these frames, the supplicant sets the authWhile timer and sends EAP-Response/Identity frames. Before the timer expires, the attacker sends the same packet again. The supplicant will repeat the same process: set the timer and send out EAP-Response/Identity frame. As a result, the supplicant is kept in the same state and the authenticator still considers the supplicant up and running. [12]

c. Lack of Authenticity

The lack of authenticity of the 802.1X frames is one of the main reasons for the session hijacking attack. Since the management frames are not authenticated, any attacker can trick the supplicant with little effort.

d. One Way Authentication

Although the 802.1X is using a controlled port mechanism for the access point, it is obvious that the port of the supplicant is always in a controlled state. The port of the supplicant is open to any entities that can pose as the access point. This is one-way

authentication. Only the supplicant is authenticated to the access point, the access point is not authenticated to the supplicant.

e. Encryption

The primary weakness of the wireless networks is the WEP encryption. Even though traditional WEP encryption is weak and vulnerable, there are access points that still do not even apply WEP so the traffic flows without any encryption. For better security, dynamic re-keying of the WEP must be an inherent part of any design. This would help prevent adversaries from intruding on the networks, even though they managed to bypass the authentication scheme.

3. Discussion of the University of Maryland Paper

Vulnerabilities of the 802.1X authentication standard were covered in the paper, "An Initial Security Analysis of the IEEE 802.1X Standard," by Arunesh Mishra and William A. Arbaugh. [1]

Session hijacking attack was one of the attacks, along with the Man-In-The-Middle attack, that the paper described as possible. A session hijacking attack is said to be performed by sending disassociate packets to the authenticated supplicant from an attacker by using the authenticator's MAC address. After this initial part of the attack, the adversary can use the MAC address of the legitimate supplicant and access the network. The lack of coordination between the two state machines lets the adversary pass through the authentication mechanism of the wireless network.

After the publication of the claims of the paper on the possibility of the session hijacking attack, Orinoco and Cisco opposed the paper's assertions. The main idea of these two papers was the same: If the wireless network is not using WEP keys, the attack can be successful. However, omitting WEP and using 802.1X authentication alone is not a common application. The attack can still be successful if the access point is using a weak WEP key. The WEP key can be broken and can be set to the attacker PC after the client is disassociated. The attack becomes more difficult if dynamic WEP rekeying is used. For this thesis, by using the test-bed, all the probabilities of WEP keying are tested and the possibilities of a successful session hijacking attack are explored.

D. SUMMARY

This chapter introduces the protocols and standards of the 802.1X authentication scheme, which can be used for wireless networks. The details of the solutions that may help overcome the vulnerabilities of wireless networks are discussed.

Even though the 802.1X standard improves the security level provided by the current 802.11 standard, there are still security leaks that may result in a session hijacking attack. An attacker may disassociate the user from the network and use its session for malicious purposes.

802.11i standard is being developed by TGi and the problems of the wireless networks will be addressed in their document. Chapter V includes an analysis of the new 802.11i standard based on the 802.11i draft 3.0.

In order to perform experiments concerning the security issues of both 802.11 and 802.1X protocols, an open-source test-bed is necessary. Chapter III will cover all the aspects of building the test-bed using open-source software.

III. AN OPEN-SOURCE WIRELESS PROTOCOL TEST-BED

A. INTRODUCTION

The IEEE 802.1X standard is proposed to address some of the IEEE 802.11 security vulnerabilities. However, the 802.1X security standard is still vulnerable to some attacks, including session hijacking attacks.

For verification and analysis of primarily session hijacking attacks and other kinds of attacks against wireless local area networks (WLANs), an 802.1X test-bed was built on an IEEE 802.1b wireless LAN. This chapter explains how to build and configure the 802.1X entities described in the last chapter: the supplicant, the authenticator, and authentication server. The open-source software was used on the Linux operating system environment for availability and ease of source-code manipulation.

B. 802.1X AUTHENTICATION TEST-BED

This thesis is based on open-source software because it is easier to demonstrate a session hijacking attack and to analyze the results of the experiment. This section explains how to combine the Linux environment with the open-source software as illustrated in Figure 7.

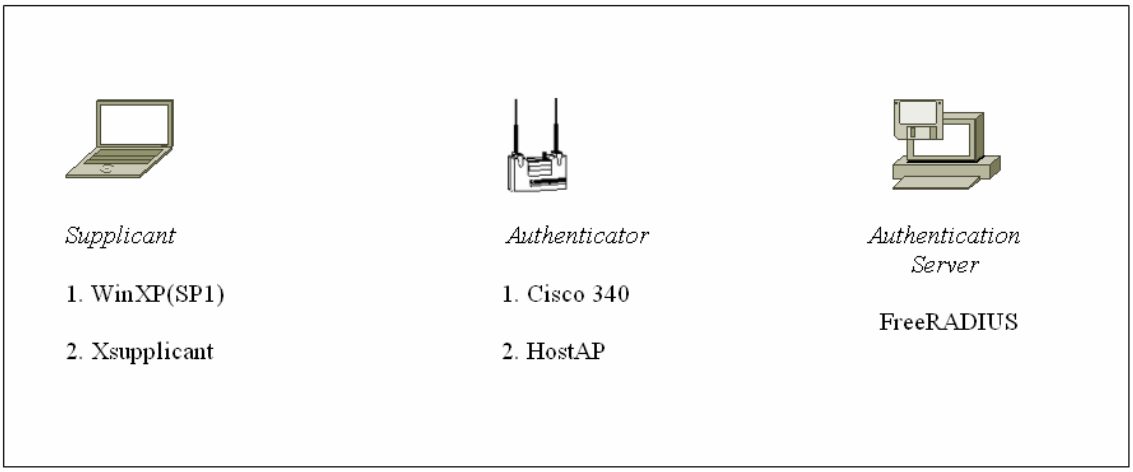


Figure 7. 802.1X Test-bed Schema (From Ref. 4)

1. Elements of the Authentication Test-Bed

The security framework of the 802.1X standard [12] consists of three main entities: supplicant, authenticator and authentication server. The roles and functionality of these three entities are explained in chapter II.

2. Authentication Methods

While building the test-bed, the most secure authentication method must be selected. One of the main factors in choosing the right authentication method for the EAP protocol is the ability to provide mutual authentication between the 802.1X entities. This is true because mounting attacks against WLANs is more difficult if the security protocol provides robust mutual authentication.

Two types of authentication methods are used and supported by the current authentication servers. The first one is Cisco's Lightweight Extensible Authentication Protocol (LEAP); the second one is the Transport Layer Security (TLS). The EAP-TLS authentication method was chosen for mutual authentication. TLS requires a public key infrastructure (PKI) for certificate-based authentication.

3. Hardware and Software Configuration of the Test-Bed

There are three entities implemented in the test-bed to support the 802.1X authentication mechanism: the supplicant (mobile client), the authenticator (access point) and the authentication server (FreeRADIUS).

a. Supplicant

The supplicant is an entity that requests network access and is being authenticated by an authenticator. A Pentium III laptop with PCMCIA support is used as the supplicant for the Open-1X test-bed. A D-Link DWL-650 wireless network interface card is used for wireless communication between the supplicant and the authenticator.

Two different operating systems can host the supplicant: Windows XP and Linux Red Hat. Both of them have advantages and disadvantages. The supplicant in Linux Red Hat provides a wide range of tools for the supplicant's configuration, while

the supplicant in Windows XP is easy to implement. In this section, both of the Windows XP client and the Xsupplicant will be explained in detail.

(1) Windows XP SP1: The Microsoft Windows XP with Service Pack 1 (SP1) operating system is used for its embedded IEEE 802.1X supplicant support. A D-Link DWL-650 network interface card is used for wireless communication. The driver can be easily downloaded and installed from http://support.dlink.com/. After the installation of Windows XP SP1, the public key certificate and private key of the client are created. The public key certificate (PKC) of the root certificate authority (Root-CA) is imported and installed. Appendix B gives the details on installing the certificates and configuring of the 802.1X protocol for the supplicant.

(2) Xsupplicant: The Linux Red Hat operating system can host the Xsupplicant on a mobile laptop. The same D-Link DWL-650 NIC is used as a wireless interface card. DWL-650 does not officially support Linux drivers. However, the chipset (Intersil Prism2) is supported by Linux Red Hat 8.0 via the *ornico_cs* driver.

The three dependent libraries should be built and installed before the Xsupplicant source code:

OpenSSL 0.9.7 (http://www.openssl.org/)

Libpcap 0.7.1 (http://www.tcpdump.org/)

Libnet 1.1.0 (http://www.packetfactory.net/libnet/)

All these components should be downloaded and uncompressed into the */usr/src/xsup* directory for the installation. The “readme” and “install” documents give enough information to guide the build of the libraries. The following commands are sufficient for building and installing the required libraries for the Xsupplicant.

cd /usr/src/xsup/<directory name>

. /configure

make

make install

The Xsupplicant source code can be downloaded from the sourceforge.net website (http://sourceforge.net/projects/open1x/). The tarball should be uncompressed into the */usr/src/xsup* directory. The following commands suffice for the default installation of Xsupplicant source code.

cd /usr/src/xsup/xsupplicant
. /configure –enable-full-debug
make
make install

The "enable-full-debug" flag causes critical information and run-time errors to be printed during configuration. It is crucial to use this flag in order to determine whether or not the dependent libraries are found by the Xsupplicant source code.

After installing the dependent libraries and Xsupplicant source code, the certificates created by Certificate Generator should be copied into the designated directory, as defined in the configuration file (*1x.conf*). This configuration file must be copied into the */etc/1x/* directory since the Xsupplicant daemon requires the *1x.conf* file to be in that directory by default. Finally, the Xsupplicant daemon can be activated with the following commands.

iwconfig wlan0 essid test
xsupplicant –i wlan0

b. Authenticator

The authenticator is an entity at one end of a LAN segment that facilitates authentication of the entity attached to the other end of that LAN. In this context, an authenticator forwards 802.1X frames from a supplicant to an authentication server for authentication. The authenticator provides network connectivity to the supplicant via a controlled port after a successful authentication. In order to achieve this, a dual-port model is used. Figure 8 shows the dual-port concept employed in IEEE 802.1X. The

authenticator has two ports for external access to the network that it is protecting: The Uncontrolled Port and the Controlled Port. The Uncontrolled port filters all traffic and allows only the EAP packets to pass for the authentication. After successful authentication, the supplicants can use the Controlled Port to send regular network traffic through the authenticator.

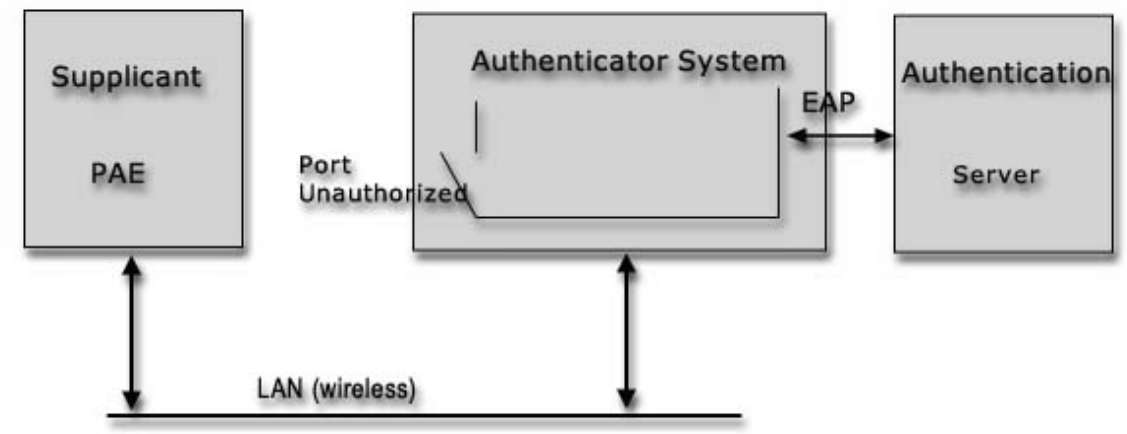


Figure 8. 802.1X Authenticator Dual Port Concept (From Ref. 4)

For this thesis, the authenticator is the most important entity in the open-source 802.1X test-bed, since it enables 802.11b access point functionality using the firmware of the Intersil chipsets for time sensitive tasks. All other functionality is handled by the HostAP driver, including WEP and passing frames off to an authentication server

(1) HostAP: The Host AP driver is a Linux driver for wireless LAN cards based on Intersil's Prism 2/2.5/3 chipset. Since D-Link DWL 650 wireless NIC is based on the Prism 2 chipset, the HostAP driver will support it. The driver supports Host AP mode and does not require any special firmware for the D-Link DWL 650 wireless NIC. It performs IEEE 802.11 management functions and acts as an access point. In addition, it implements the following IEEE 802.11 functions:

Association
Authentication
Data transmission between two wireless stations
Power saving mode signaling

Frame buffering.

An IBM Think-Pad 600 Pentium II laptop and a D-Link DWL 650 wireless NIC are the hardware components of the authenticator. Linux Red Hat 9.0 with kernel 2.4.22 is installed to support the HostAP driver.

The Wireless Extensions v15 and v16 patches are not necessary for Linux kernel version 2.4.22. Two kernel configuration options must be enabled during the configuration of the kernel (2.4.22): the wireless LAN (non-hamradio) option for HostAP support (Figure 8) and the 802.1d Ethernet Bridging option (Figure 9) for bridging support between the wireless and wired interface. These graphical configuration menus will be displayed after the *make xconfig* command of the kernel configuration process. These options are not enabled by default. The authenticator would not function properly without these support options

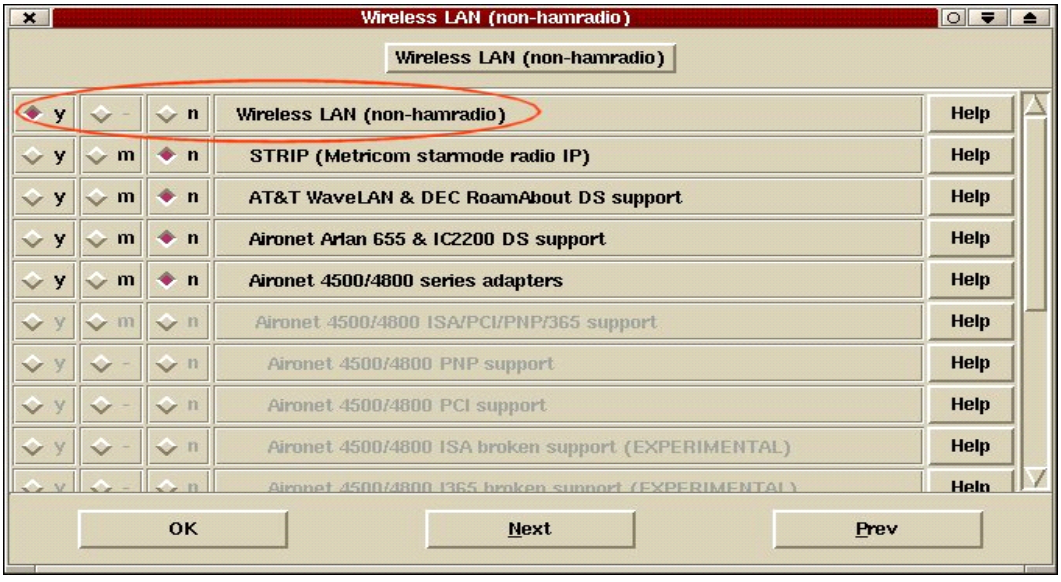


Figure 9. Wireless LAN (non-hamradio) Option

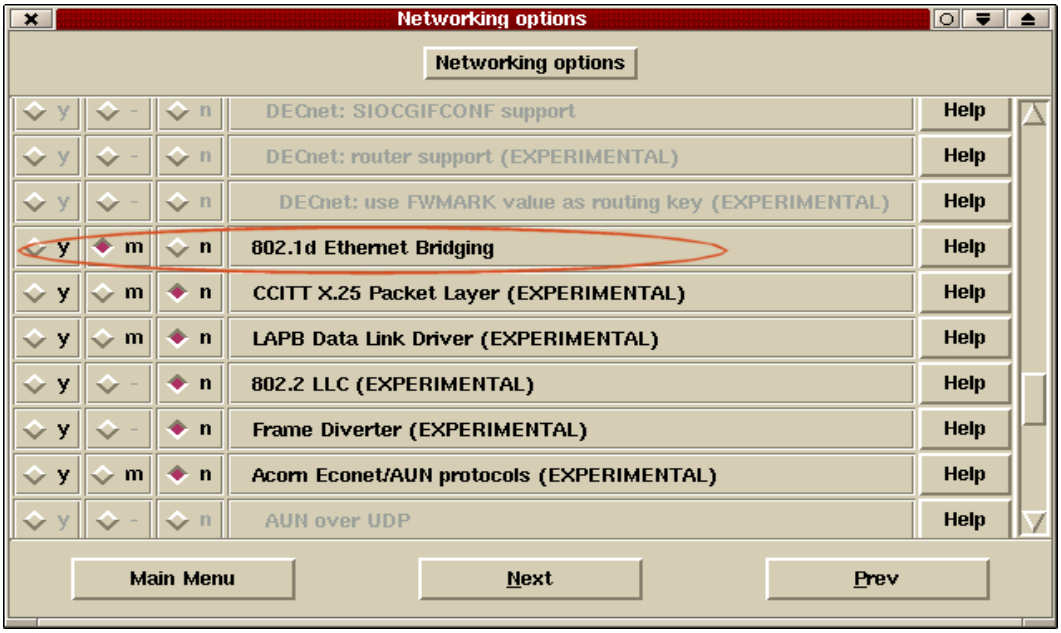


Figure 10. 802.1d Kernel Bridging Support

Before version v0.1.0, the HostAP driver used to be distributed as one tarball. Now the software is separated into three components. Since the HostAP v0.1.1 driver is used for the test-bed, the following three HostAP components and the Wireless Extension tools were installed on the authenticator:

HostAP-driver 0.1.1

Hostapd

Hostap-utils

Wireless Extension Tools v25

The Wireless Extensions Tools v25 (http://pcmcia-cs.sourceforge.net/ftp/contrib/) were downloaded and uncompressed into the */usr/src/* directory tree. The source code was installed by the command sequence below.

. /configure

make

make install

The HostAP driver, utilities and daemon source code (http://hostap.epitest.fi) were downloaded and uncompressed into the */usr/src/* directory tree. The HostAP driver was configured by the commands below.

tar –zxvf hostap-driver-0.1.1.tar.gz
. /configure

After the configuration of the driver, a *Makefile* was created for this specific configuration. The KERNEL_PATH variable was set to the kernel configured for the access point in the */usr/src/hostap/Makefile* file.

 # Edit this path to match the system (It should point to the root
 # directory of the Linux kernel source.)
KERNEL_PATH=/usr/src/linux-2.4.22
 # Leave this blank for kernel-tree PCMCIA compilations
PCMCIA_PATH=

Since the HostAP requires kernel support, the HostAP source code must be built and installed after the proper configuration and required modification to the *Makefile.* The "Extra flag" option was required in order to support the 802.1X functionality for hostapd daemon.

make pccard EXTRA_CFLAGS="-DPRISM_HOSTAPD"
make install_pccard

The HostAP utility and daemon components were built and installed to support the 802.1X authenticator functionality with the following sequence of commands:

configure
make
make install

The authenticator supports bridging since the 802.1d Ethernet Bridging option was checked during the kernel configuration. Bridging will provide communication between the wireless and wired segments of the network. Otherwise, the authenticator cannot relay the regular network packets between the supplicant and the network. There must be two interfaces in the authenticator: an Ethernet interface (eth0) connecting the wireless segment to the wired network and a Wireless interface (wlan0) acting as an access points. The following series of commands are used to establish the bridging between the two interfaces.

ifconfig wlan0 0.0.0.0
ifconfig eth0 0.0.0.0
brctl addbr br0
brctl addif br0 eth0
brctl addif br0 wlan0
ifconfig br0 XXX.XXX.XXX.XXX up

Both interfaces' IP addresses must be set to zero and assigned to the bridge interface as defined above. The bridge interface (br0) is a logical interface rather than a physical one like eth0 or wlan0. The AP bridges packets between the Ethernet and wireless LANs and can be reached with the IP address XXX.XXX.XXX.XXX from either network. When the AP reboots, the bridging between the interfaces should also be reestablished.

After installing all the components and establishing the bridging, the authenticator was ready to run the *hostapd* daemon to serve as an 802.1X compliant access point. The *hostapd* daemon accesses a configuration file known as *hostapd.conf* to retrieve the parameters of the wireless network. Appendix C provides an example of this configuration file. The *hostapd* daemon is launched with the following command sequence.

cd /usr/src/hostap/hostapd
. /hostapd –d hostapd.conf

(2) D-Link DWL-7000AP: Since this Access point supports the 802.1X authentication protocol, it was chosen to be used as the authenticator of the test-bed. The configuration of the access point was simple since it has a web-based configuration tool. Appendix C explains and illustrates the configuration of DWL-7000AP.

c. Authentication Server

Since FreeRADIUS (http://www.freeradius.org/) is the only available open-source authentication server tool that supports Linux, it was used by the test-bed for this thesis. FreeRADIUS supports the EAP-TLS authentication method as an embedded module.

One of the latest versions of the Red Hat Linux (Red Hat 8.0) was used as the operating system for the authentication server. The newer versions of FreeRADIUS are compatible with Linux Red Hat 8.0. In order to run the FreeRADIUS server on Linux Red Hat 8.0, the following software should be downloaded and installed.

OPENSSL 0.9.7

OPENSSL SNAP-20020227

OPENSSL 0.9.7-beta3

FREERADIUS-0.9.2

Three different versions of the OpenSSL are needed throughout the whole process. The stable version of OpenSSL (OPENSSL 0.9.7) is required to build most of the FreeRADIUS software. A recent snapshot version of OpenSSL (OPENSSL SNAP-20020227) is required to build the EAP/TLS modules. OpenSSL 0.9.7-beta3 is the third version of the OpenSSL, which is used to create the certificates.

The other open-source software package that must be installed is FreeRADIUS-0.9.2. After successfully installing all of the software, the Linux computer becomes the authentication server of the test-bed. The installation procedures of the

necessary software come with the downloadable tarball. The following paragraphs emphasize the important details of the installation process.

(1) OPENSSL 0.9.7: OpenSSL 0.9.7 (http://www.openssl. org) is used for building FreeRADIUS. After downloading and uncompressing the tarball, the following sequence of commands is used to build the software.

cd openssl-0.9.7
. /config
make
make test
make install

These commands build and install the OpenSSL-0.9.7 in the default location, which is */usr/local/ssl* for Linux Red Hat 8.0. For the test-bed, the default location was used.

(2) OPENSSL SNAP-20020227: The snapshot version of the OpenSSL is used to load the EAP-TLS module in the FreeRADIUS source code. After downloading and uncompressing the OpenSSL 0.9.7 tarball, the source code was built and installed by using the commands below:

. /config –prefix=/usr/local/openssl shared
make
make test
make install

One of the most important parts of this installation is paying attention to the location where the software is installed. If the config command was used without any switches, this version of OpenSSL would be installed to the default location and overwrite the stable version previously installed.

The final part of the installation is checking and verifying that *libssl.so* and *libssl.so.0* are symbolically linked to *libssl.so.0.9.8* while *libcrypto.so* and *libcrypto.so.0* are symbolically linked to *librypto.so.0.9.8*. These files can be found under the *lib* directory of the snapshot version of OpenSSL.

(3) OPENSSL 0.9.7-beta3: This version of the OpenSSL is necessary to generate the certificates used by the wireless network. This software can be installed on another computer and the certificates can be generated in a more secure location in real-life applications. However, to keep the process simple, the authentication server computer was used as the certificate generator as well. After downloading and uncompressing the software, the following commands were used to install this version of OpenSSL:

. /config –prefix=/usr/local/openssl-certgen shared
make
make test
make install

The final step of the installation is checking and verifying the same sym links of the specific lib files, which were mentioned in the installation of the SNAP version. These files can be found under the *lib* directory of the beta version of OpenSSL. Appendix A provides the OpenSSL configuration file for the certificate generator.

(4) FREERADIUS-0.9.2: The latest version of FreeRADIUS (FreeRADIUS 0.9.2) modules were downloaded and uncompressed into the */root/downloads* directory. The FreeRADIUS source code was configured before building by using the commands below:

cd /root/downloads/freeradius-0.9.2
. /configure –sysconfdir=/etc

The configuration script prepares the *makefile* to build the source code. Since the OpenSSL version that will be used to create the EAP-TLS modules is different than the default OpenSSL, The EAP-TLS *makefile*, which was placed in */root/downloads/freeradius-0.9.2/src/modules/rlm_eap/types/rlm_eap_tls/* directory, was modified as defined in Appendix D. After modifying the *makefile*, the FreeRADIUS can be installed by using the following commands:

make
make install

After building and installing the FreeRADIUS software, the configuration files of the RADIUS server should be modified to enable the 802.1X authentication scheme. There are three configuration files that should be modified: *radiusd.conf, users,* and *clients.conf.* The *radiusd.conf* file is modified to make EAP-TLS work properly. The c*lients.conf* is modified to allow access by the access points of the wireless network to request authentication. The *users* file is modified to include pointers to client certificates. A test user may also be temporarily added to the *users* file to check the functionality of the authentication server without using the other components. Appendix D provides a copy of all three configuration files.

For the session-key management, two random files must be created. These files only need to contain random characters. For this particular occasion, the *date* command was used to create these two random files.

date > /etc/1x/random
date > /etc/1x/DH

If everything has been installed and configured correctly, the FreeRADIUS should be ready to run and to authenticate the legitimate users via the EAP-TLS. A wrapper script is created to run FreeRADIUS with the correct SSL libraries of the OpenSSL snapshot version. Appendix D provides a copy of this script (*run-radiusd*).

The FreeRADIUS may be run from the shell by typing *run-radiusd –X –A*. (This runs the script that is mentioned in the preceding paragraph). After seeing that the server is running correctly and listening for requests, the test user can be employed to test the server. If the result of the test is good and an "Access-Accept" message is returned, then the server is running well and the test user may be deleted.

4. EAP-TLS Authentication Method

This section discusses the Extensible Authentication Protocol Transport Layer Security (EAP-TLS) briefly since EAP-TLS protocol was examined in [4] in detail.

EAP-TLS authentication is based on 802.1X/EAP architecture. The three entities involved in the 802.1X/EAP authentication process are supplicant (the end entity), the authenticator (the access point), and the authentication server (RADIUS server). The supplicant and the RADIUS server must support EAP-TLS authentication. The access point has to support the 802.1X/EAP authentication process. For example, a Cisco Aironet access point that supports the EAP-TLS authentication protocol can be used in the 802.1X test-bed.

The 802.1X test-bed requires the EAP-TLS authentication protocol for mutual authentication and key exchange between the entities. EAP-TLS (RFC-2716) uses the TLS protocol (RFC-2246), which is the latest version of the Secure Socket Layer (SSL) protocol. TLS provides a mechanism for both user and server authentication and for dynamic session key generation in order to use certificates. Figure 11 illustrates the general schema for a message exchange between the entities.

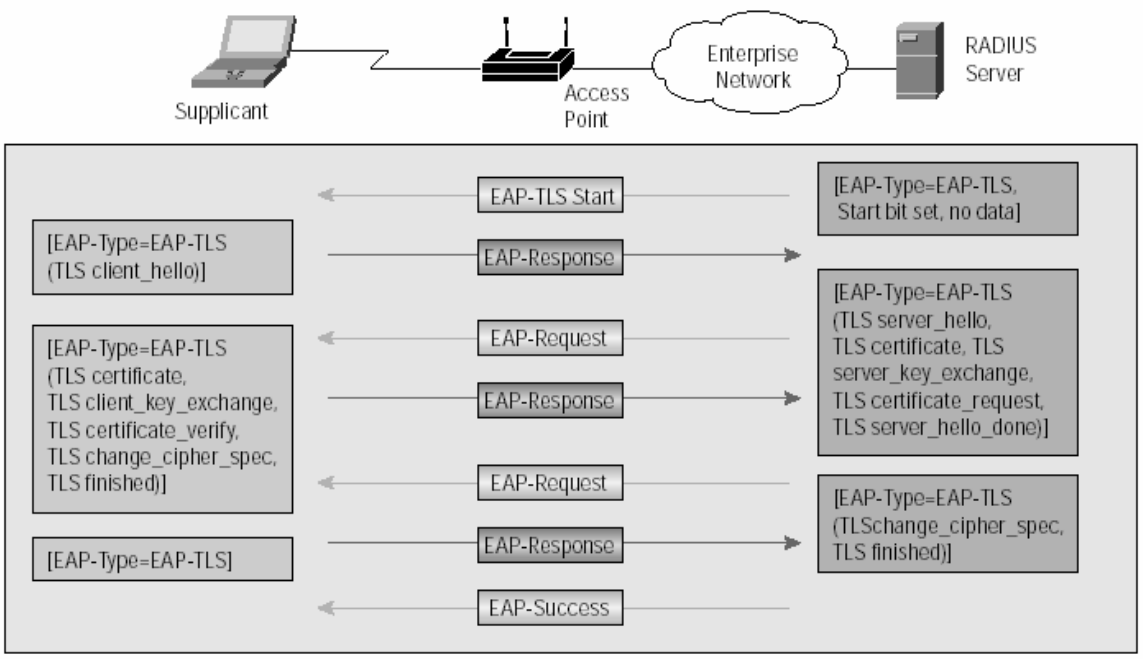


Figure 11. EAP-TLS Message Exchange (From Ref. 26)

5. X.509v3 Certificates

The EAP-TLS authentication method is certificate-based. This method uses X.509v3 certificates, which can be generated by using the OpenSSL software. Appendix A covers the OpenSSL configuration file of the certificate generator.

X.509v3 certificates contain the public key of the supplicant with some additional information. The certificates are created for the Root Certificate Authority, authentication server and the supplicant. The Root CA certificate is created first; the other certificates are digitally signed by the private key of the CA.

Three certificate generation scripts will be used to create all the necessary certificates. Appendix B provides a copy of all three certificate generation scripts with an XP specific extension file that is necessary to generate a certificate. One of the most important requirements when creating the certificates is to verify that the client name on the certificate matches the names in the users configuration file. Another critical point is to assign a password that is different from the default password (default is “whatever”) during the certificate generation process.

The first certificate to be generated is the root certificate. This certificate will be used to sign the other two certificates. So the generation sequence will be as follows:

./CA.root

./CA.srv <servername>

./CA.clt <clientname>

After all three scripts are run, the folder will contain 12 different certificates. The extensions of the certificates will be ".p12, .der, .pem". The two certificates that will be used by the FreeRADIUS are <servername>.pem and root.pem. The WinXP supplicant client requires <clientname>.p12 and root.der. So those two certificates should be copied to the supplicant and installed. The details of the certificate generation process can be found on the following web sites:

http://www.impossiblereflex.com/8021x/eap-tls-HOWTO.htm

http:// www.missl.cs.umd.edu/ wireless/eaptls/

6. Validation

After installing all the software to run the test-bed, the system was tested by using the packet-capture software, Ethereal. (www.ethereal.com). Ethereal is a software tool that captures all the network traffic. The user may select the network adapter to filter the traffic. Two laptops were used in the validation process. One of the laptops (Dell Insiron-5100) was configured to capture the wireless packets in promiscuous mode with Ethereal. The other laptop (HP Compaq pavilion Ze5400) was used as the supplicant.

The client certificates were installed to the supplicant, and the wireless card was enabled to initiate the authentication process. All the traffic concerning the authentication process was captured by Ethereal which was running on the other laptop. The packets captured by Ethereal, shown in Figure 12, verified a successful authentication process.

Appendix E provides the successful logs of both the authentication server and the authenticator.

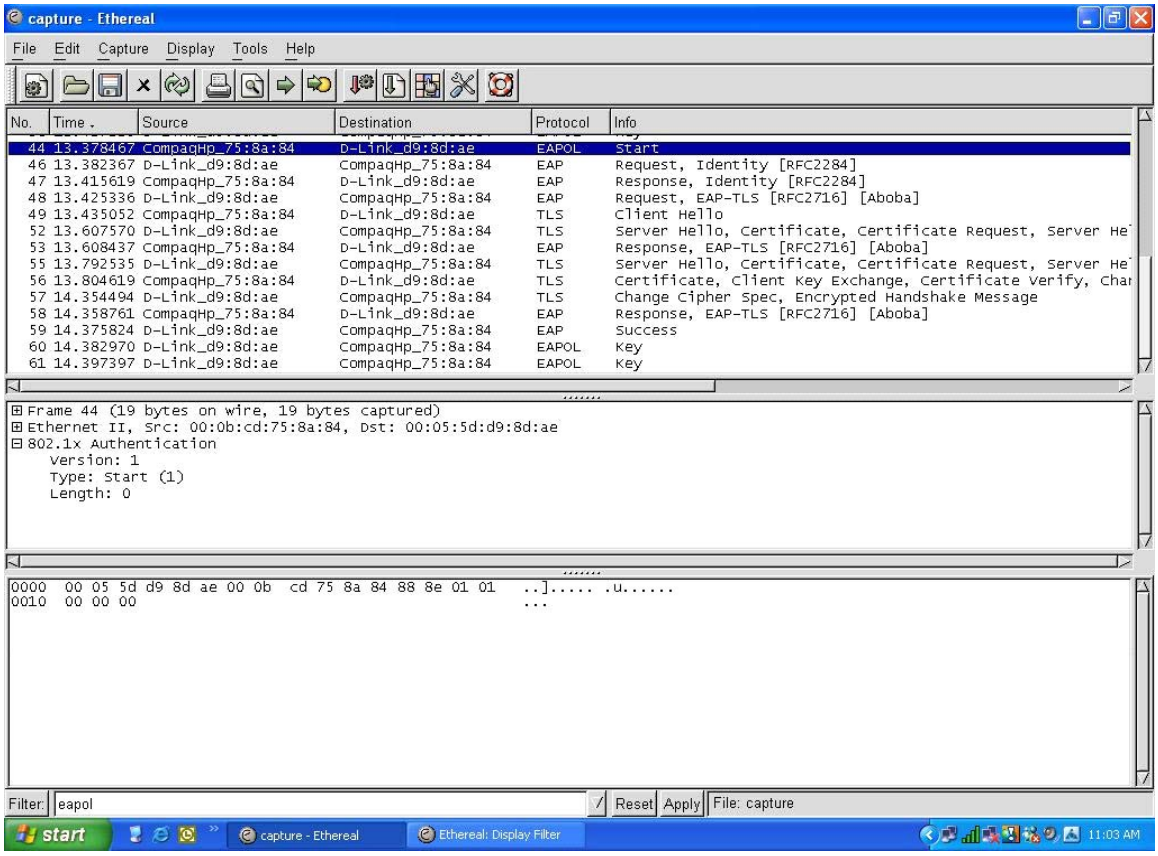


Figure 12. Captured Packets showing a successful authentication

C. SUMMARY

This chapter covers the building of the open-source test-bed. The test-bed is built using the open-source software and the Linux operating system. 802.1X authentication standard is applied with a specific certificate-based authentication method: EAP-TLS.

A detailed manual about the configuration and installation of the 802.1X test-bed is provided in this chapter. The experiments and analysis of the session hijacking attack will be presented in the next chapters.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. APPLICATION OF THE SESSION HIJACKING ATTACK

A. INTRODUCTION

This chapter describes our first-hand experience of implementing a session hijacking attack against a WLAN. To implement an attack of this nature, an open-source platform should be built. The previous chapters detailed the building of such an open-source test-bed. The test-bed was built to reflect a real-life implementation of a wireless network that uses the 802.1X authentication protocol.

A session hijacking attack needs a laptop that can serve both as a rogue access point and as a rogue client. The test-bed works with a commercial access point (D-Link DWL-7000AP), which is capable of 802.1X authentication. A laptop with the HostAP software is used as the rogue access point for the attack described in this chapter. For the rogue client, either the same HostAP laptop or another laptop with Windows XP running is used during the implementation of the attack.

The University of Maryland paper [1] talked about the possibility of a session hijack attack and the resultant responses by Cisco and Orinoco acknowledged the same risk, however, with some exceptions. Our work provides additional observations from a practical perspective.

B. NECESSARY CONDITIONS FOR SESSION HIJACKING ATTACK

A session hijacking attack is based on some specific vulnerabilities that can be exploited in such a fashion that both legitimate entities of the session are kept unaware of the attack. The nature of the attack involves posing as the authenticated user and using that user's credentials to use the protected network.

The first specification of the 802.11 standard is wide open to any kind of attacks. The standard does not even mandate the use of encryption for network traffic and authentication for the users. Using the 802.11 standard without encryption makes it subject to snooping and hijacking, regardless of the authentication method used.

The enhancement to the first specifications of the 802.11 standard resulted in the introduction of the popular and widely discussed encryption technique, known as Wired

Equivalence Privacy (WEP). The weaknesses in the WEP encryption were discovered shortly after its introduction. WEP does not support per-packet integrity and offers weak encryption. The length of the initialization Vector (IV) is short enough (24 bits) for an attacker to break the WEP key. In a congested wireless network, an attacker can monitor the network traffic and capture multiple packets encrypted with the same WEP key and same IV after a fairly short time. The WEP key is then broken easily.

There are two main conditions that should be present for a general session hijacking attack: the server should be unaware of the attack and the client should be unaware of the attack. The other two conditions mentioned below are the specific conditions that should be met for a specific session hijacking attack against a wireless network.

From a mathematical point of view the attack can be formulated as follows:

T_0: the time of the start of the attack

T_1: the time of the disassociation of the client

T_2: the time that the attacker takes over the client's role after changing the settings of the attacker

T_3: the time of the arrival of the first replay packet for the malicious packets.

T_4: the time of the detection of the attack by the network administrators.

T_e: time to break the encryption key.

Since the T_e is much larger than the other time variables of the attack, the best practice for an attacker is to break the key before the actual start of the attack. The time necessary for a session hijacking attack to be complete ($T_{complete}$) can be formulated as follows:

$$T_{complete} = T_3 - T_0 = (T_3 - T_2) + (T_2 - T_1) + (T_1 - T_0)$$

The time of the detection of the attack (T_4) cannot be smaller than T_1. The attack can be detected as soon as the client is disassociated. Thus,

$$(T_4 >= T_1).$$

For a session hijacking attack to be evaluated as complete and successful, the following equation should be met:

$T_{complete} < T_4$

$(T_3 – T_2) + (T_2 – T_1) < T_4$

This formulation shows that the attack can only be successful if the attack is not detected before the reply for the first malicious packet returns to the attacker.

Following are the necessary conditions specific to a session hijacking attack in a wireless network.

1. Necessary Condition 1:

For a session hijacking attack, the server or the entities that are responsible for providing network services should be kept unaware of the existence of an attacker for a sufficient amount of time, as described in the aforementioned formulation.

2. Necessary Condition 2:

Similar to the server, the legitimate client that is disassociated from the network should also be kept unaware of the existence of the attacker.

The legitimate client is both the entity that will suffer from the attack and the entity that can detect the attack. The attacker will keep the legitimate client out of the network. If the legitimate client becomes aware of an attacker, the precaution against the attack may be applied. The time for the client to become aware of the attack is not controlled by the attacker. This time varies according to the instincts and knowledge of the client. A careful and knowledgeable client can detect an attack just after the attack is initialized.

3. Necessary Condition 3:

For a session hijacking attack, the attacker should have all the necessary tools and equipment that may be necessary at any part of the attack.

The steps of the session hijacking attack will be discussed in detail in this chapter. The attacker should have equipment that has good peak power to challenge the legitimate entities. Once the power of the attacker is greater than the legitimate entity, the victim machine will try to communicate with the station with the higher power.

The attacker should have tools, such as Netstumbler and Ethereal, to monitor the victim's network. Gathering information about the victim's network is the first step for launching an attack. The attacker should have the capability to run a WEP key breaker tool against the victim network. The attacker should also be able to create malicious packets that look as if they are coming from a legitimate user. The tools that are used to generate the packets are discussed later in this chapter. The attacker should also have the capabilities and knowledge to change and to spoof IP and MAC addresses.

4. Necessary Condition 4:

No encryption or weak encryption is in use by the target network.

A session hijacking attack is possible with any kind of known authentication mechanisms, unless encryption is used in the network. The 802.1X authentication mechanism provides some good techniques for authentication. EAP-TLS is one of the most popular authentication mechanisms. EAP-TLS provides strong mutual authentication, however, due to the problems discussed in Chapter II, the session hijacking attack is still possible.

The importance of the encryption is mentioned in a paper published by Cisco [2] as an answer to the University of Maryland's paper [1]:

> If the network uses an EAP authentication algorithm that does not support dynamic WEP keys of mutual authentication, the wireless LAN will be vulnerable to attack. An example is the use of the EAP-MD5 authentication algorithm.
>
> EAP- MD5 performs one way authentication of the client with no facility for dynamic WEP (static WEP is supported)…

Another response published by Orinoco [3] accepts the possibility of a session hijacking attack without using encryption:

The hijacked session attack assumes that no encryption is present. When no encryption is present, this attack will succeed, allowing the attacker to use the session until the next re-authentication interval. At the next re-authentication time, the attacker would not be re-authenticated. He would then hijack another valid session.

The use of the static WEP key may not be sufficient to prevent all the attempts of the session hijacking. If the same WEP key is being used by all the clients of the network, that WEP key is vulnerable to compromise. The length of the WEP key and the traffic load of the traffic using it are important factors that effect the time needed to break the WEP Key.

Dynamic, session-based, per user WEP keys will be a good solution to prevent the attackers to use the network, even if they manage to steal the session. The session hijacking attack may devolve to a denial of service attack for the legitimate client if the Dynamic WEP keys are used.

C. DEMONSTRATION OF SESSION HIJACKING ATTACK

As mentioned earlier in this thesis, a session hijacking attack is possible because clear communication between the state machines of the authenticator and the authentication server is lacking. A. Mishra and W. Arbaugh claim that the IEEE 802.1X is vulnerable to a session hijacking attack in their paper, "An Initial Security Analysis of the IEEE 802.1X Standard" [1]. They mentioned that after a successful authentication by the legitimate client, the attacker can send disassociate frames to the client by spoofing the legitimate authenticator and as a result, the supplicant is disassociated from the network. After the first part of the attack is completed, the attacker can spoof the MAC address of the legitimate supplicant and gain access to the network without passing the authentication scheme. The actual attack consists of three parts;

- The disassociation of the supplicant
- Breaking the encryption if encryption is in use.
- Accessing the network by using the credentials of the disconnected user.

1. Disassociation of the Supplicant

The first two necessary conditions, the server and the legitimate client are kept unaware of the attack, must be present to exercise this step.. Condition 3 is necessary to complete this step, also. The attacker must monitor the network using the type of tools described in the following chapters.

The attacker has two machines to launch the attacks. Both machines are used to monitor the network traffic and the MAC addresses of the entities. Netstumbler is one of the Windows-based tools that may be used to detect he wireless access points (Figure 13). Windows XP also has the capability to catch the available wireless networks. Netstumbler is used to monitor the SSID of the network and the MAC addresses of the authenticator.

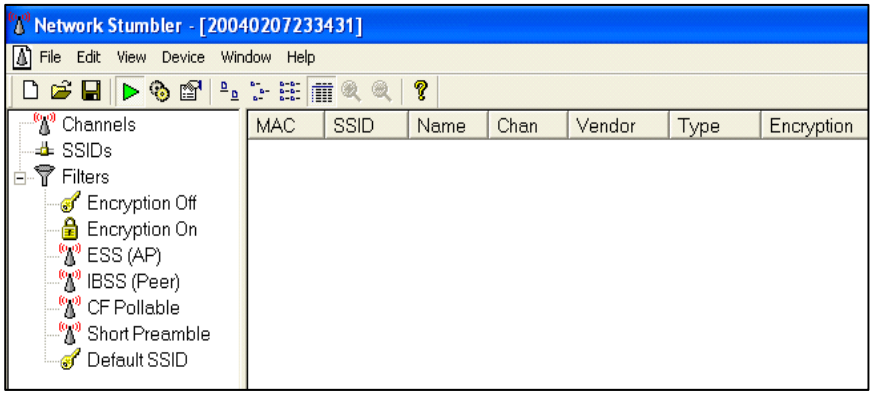


Figure 13. Netstumbler

The second important tool used to monitor the network traffic is Ethereal. Ethereal can work in both Linux and the Windows operating systems. Using Ethereal, an attacker can monitor all the wireless traffic in promiscuous mode. A successful authentication message sequence between the legitimate supplicant and the authenticator was monitored by using Ethereal, as shown in Figure 14. The messages exchanged and the MAC addresses of the entities are shown.

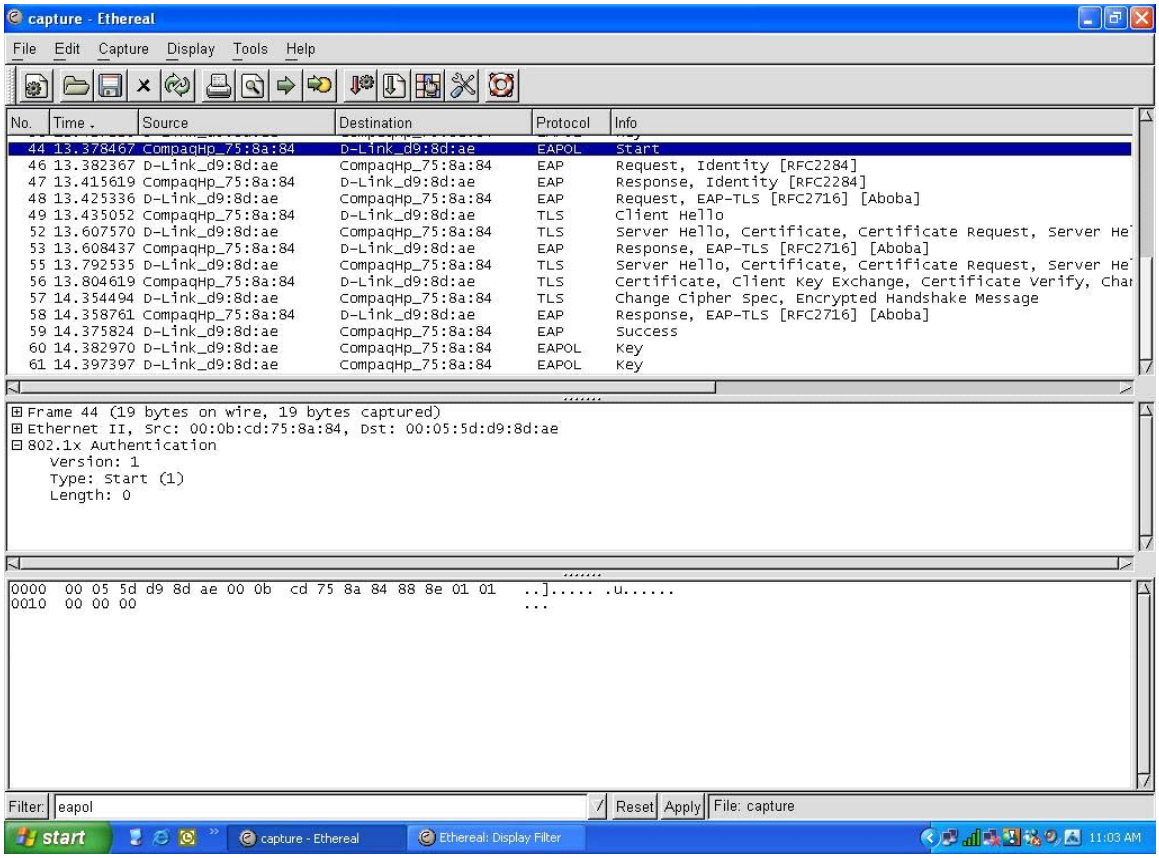


Figure 14. Successful Authentication Traffic Captured by Ethereal

The MAC address of the rogue access point (HostAP) should be changed to the MAC address of the legitimate access point. This is relatively simple in the Linux environment compared to the Windows OS. Since the Hostap driver is working properly in the rogue access point, the ifconfig command easily changes the MAC address and the SSID of the wireless interface.

ifconfig wlan0 hw ether <MAC address>

iwconfig wlan0 essid "test"

The preceding two lines of commands change the MAC address of the rogue access point. After the capture of the MAC address, there are two ways to send disassociate the legitimate supplicant. The first one is to run the HostAP software with the following command:

./hostapd –d hostap.conf

HostAP automatically creates disassociation packets each time it is initiated. Since the supplicant just checks the MAC address of the frames coming from the access point, it is disconnected after the packet is received. The second way of disassociation is using the wireless tools that come with the HostAP tarball. The following command is used to disconnect the authenticated client:

iwpriv wlan0 kickmac <MAC address>

However, the station will try to authenticate and associate again immediately after this, so the user should be denied by the rogue access point using the following commands:

iwpriv wlan0 maccmd 2

iwpriv wlan0 addmac <MAC address>

2. Breaking the WEP Key

Necessary condition 4 is critical for this step. If the encryption is not used, then there is no need to break the WEP key, and the access to the victim network is trivial. If the necessary condition 4 is not met, then the attack is unsuccessful.

WEP (Wired Equivalent Privacy) is the encryption technique used by the 802.11 wireless networks. Since Wireless LANs can be accessed without physical connection to the LAN infrastructure, IEEE decided to use encryption at a Data Link Layer to prevent eavesdropping. There have been several discussions and papers about the vulnerabilities of WEP. These vulnerabilities will be mentioned, since session hijacking attacks may include breaking the encryption of the wireless traffic.

WEP is on the RC4 algorithm, which is a symmetric key stream cipher. Stream cipher algorithms generate a key stream from the original key to match the length of the plaintext that will be encrypted. The stream ciphers, along with another method called "block-cipher" are known as Electronic Code Book (ECB) mode encryption. With ECB mode encryption, the same cipher text is generated when the input plain text is the same. Since usage of the same key for each packet would cause security problems, the Initialization Vector (IV) is used to obtain a different key for each packet by using the

same Key. By using the IV, the resulting cipher text will be different, even if the same packet is transmitted twice.

An Initialization Vector is 24 bits long and is sent in clear to the recipient of the packet. The WEP key is 40 or 104 bits and with the augmentation of the IV they become 64 or 128 bits long. Although the length of the WEP key is known to be 64 or 128 bits, it is actually 40 or 108 bits, since the IV is sent in the clear and readily available to the eavesdroppers.

The weakness of the WEP encryption algorithm was revealed by two studies in 2001: "Intercepting Mobile Communications: The Insecurity of 802.11" [15] and "Weakness in Key Scheduling Algorithm of RC4" [10]. Cryptanalysts Fluhrer, Mantin, and Shamir determined that WEP key can be broken passively by just collecting packets from a wireless network [10]. This technique is based on weak IVs, which reveal the key bytes after statistical analyses. This vulnerability was implemented by AT&T/Rice University and the developers of an open-source WEP cracking tool, Airsnort. The implementation showed that 64 or 128 bits long WEP keys may be derived after analyzing as few as four million packets [20]. This amount of packets is routed in four hours in a congested wireless networks. A 24–bit IV can contain 16,777,216 possible values. So, in a network running at 11 Mbps, the time for the same IV to be used is about five hours. [21]

WEP specifications do not require any specific kind of key management technique. The first of three techniques to be used for key management is static WEP keying. The shared secret key is known by both of the entities and that key is used without any change. The second key management technique is creating a known number of WEP keys and using those keys interchangeably. The third one is dynamic WEP rekeying, in which the new WEP keys are produced and installed automatically at small intervals without the supervision of any system administration.

A wide range of tools was created after the vulnerabilities of the WEP algorithm became public. Airsnort and WEPcrack are the two most popular tools. WEPcrack is a series of perl scripts that are used to break the WEP key of the collected packets by a sniffer. Airsnort does not require a separate sniffer, since Airsnort can both capture the

packets and crack the WEP key. For this thesis work, Airsnort was selected as the WEP cracking tool.

Airsnort is a Linux-based tool that exploits the vulnerabilities mentioned in the paper "Using Fluhrer, Mantin and Shamir Attack to Break WEP" [11]. Arisnort requires Linux kernel 2.2 or 2.4 with the wlan-ng drivers and a prism2-chipset-based wireless network card. The difficulty of the open-source wireless tools is to match the combination of the kernel configuration with the proper driver and the networking card. Airsnort works perfectly after overcoming the difficulties of the installation. Airsnort has a GUI interface which is sufficient for any user to interact with the software. The GUI interface is shown in Figure 15. The Cracking time depends on both the length of the WEP key and the amount of network traffic. The amount of time could vary, but it is obvious that with one of the tools and patience, the WEP keys can be broken.

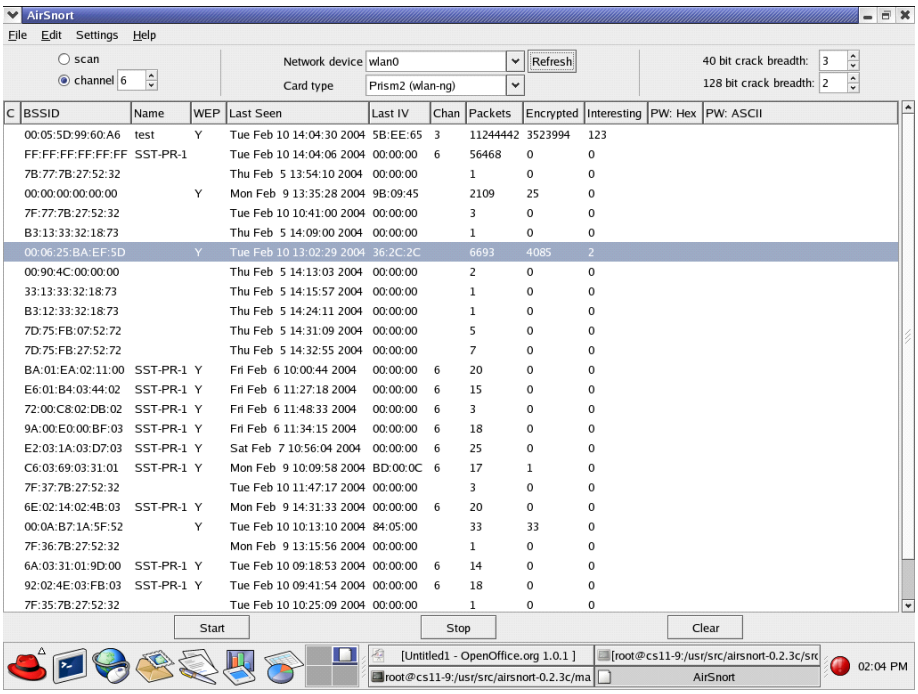


Figure 15. WEP Cracking Tool Airsnort

3. Accessing the Network

After completing the first two steps of the session hijacking attack, there is only one step left to complete the attack. The last part is accessing the victim network by using the victim client's credentials.

The victim client will be out of the picture at the end of the first part. However, the client will try to reconnect to the network after it is disconnected. There will not be enough time for the second part of the attack if encryption is used. Therefore, breaking the WEP key should be done prior to the first step of the attack.

If the network is not using encryption, the attacker can change its MAC address and use the network as the real client. The attacker learns the MAC address of a legitimate client after the first step of the attack. If the attacker is using a Linux machine, changing the MAC address is the same as the first part. If the attacker is using Windows, then a tool must be used to change the MAC of the wireless interface. There are some free tools that can be used for this purpose. One of the most popular is SMAC, developed by KLC Consulting Inc. That tool can change the MAC addresses for Windows compatible NICs (http://www.klcconsulting.net/smac/). This tool has a GUI interface and is easy to use. There is one drawback to this tool: the machine should be rebooted for the change to take effect. Figure 16 shows the interface of SMAC.

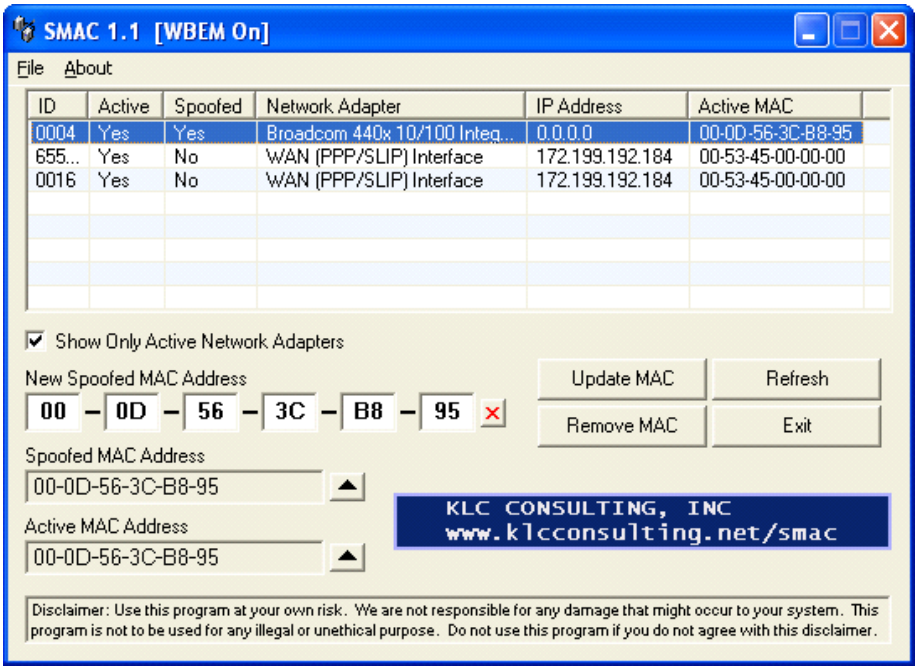


Figure 16. SMAC: MAC Changer Tool Interface

After changing the MAC address, the Windows XP client should configure the wireless connection options to connect to the network. There are some options related to

the connection properties, namely the use of encryption and the size and type of the encryption Key.

1. No Encryption: If there is no encryption used, the settings should be made for a connection without an encryption using the GUI interface for Windows XP wireless connection settings. Figure 17 shows the connection settings for a connection without encryption.

Theoretically, the Windows client should continue on the connection that is provided by the previously connected legitimate client. For test purposes, the commercial access point (D-Link DWL-7000AP) was the first choice to be configured to use the 802.1X authentication without encryption. However, the access point software doesn't allow this kind of setup. The access point allows the 802.1X authentication only with encryption enabled. The other access point used in this thesis, HostAP, does allow any kind of configuration since it is an open–source tool. The configuration file called *hostapd.conf* can be configured to be used without encryption.

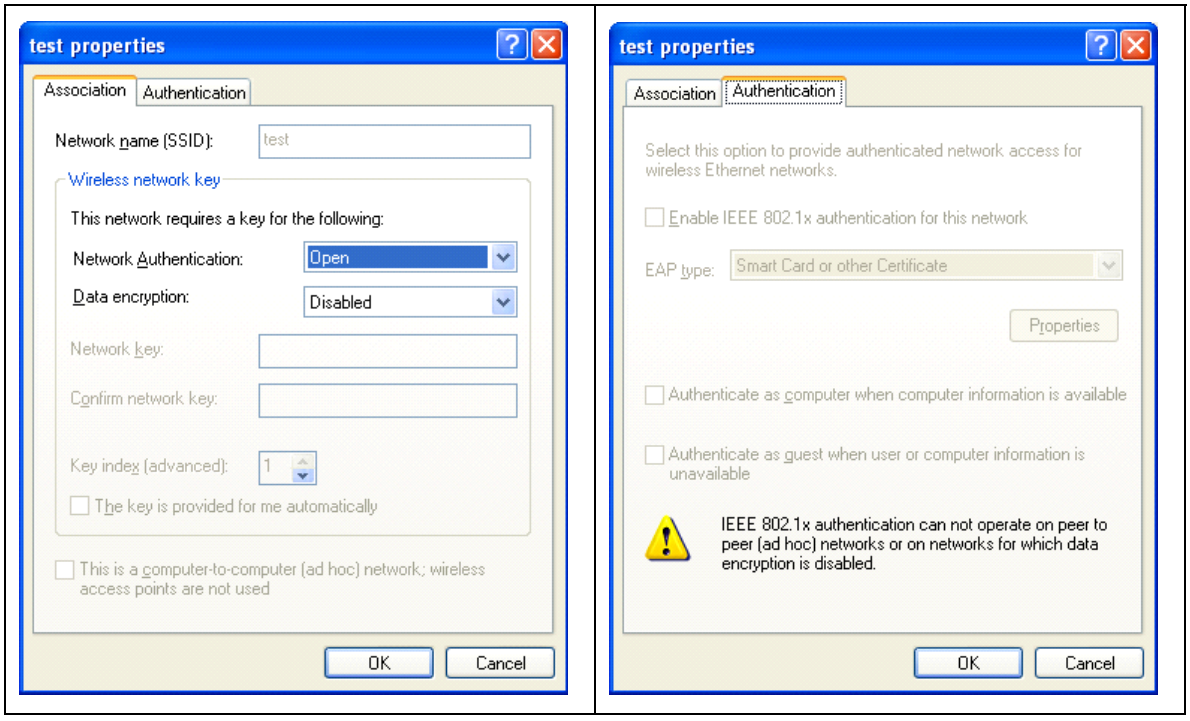


Figure 17. Windows XP Wireless Configuration Setup without Encryption.

2. With Static WEP key encryption: When static encryption is used, with 64- or 128-bit keys, the XP client can be configured to use a connection similar to the original connection with encryption. The static WEP keys are known as breakable with tools like Airsnort. The tool was tested and left to break the 64 bit key of the test-bed. For four days of action, the tool collected only 123 interesting packets, whereas at least 1500 interesting packets are necessary to break the key. The tool could not break the key in four days since the network traffic of the test bed was not as high as expected. Although the WEP key could not be broken successfully due to the time considerations, the key is assumed to be broken by the tool since WEP cracking is not the main issue of this thesis.

Both D-Link and HostAP access points are easy to be configured to work with static WEP keys and the 802.1X authentication. After the configuration of the XP client with the known WEP key, the XP client can use the network by activating its network card. Figure 18 shows the configuration options with the static WEP key.

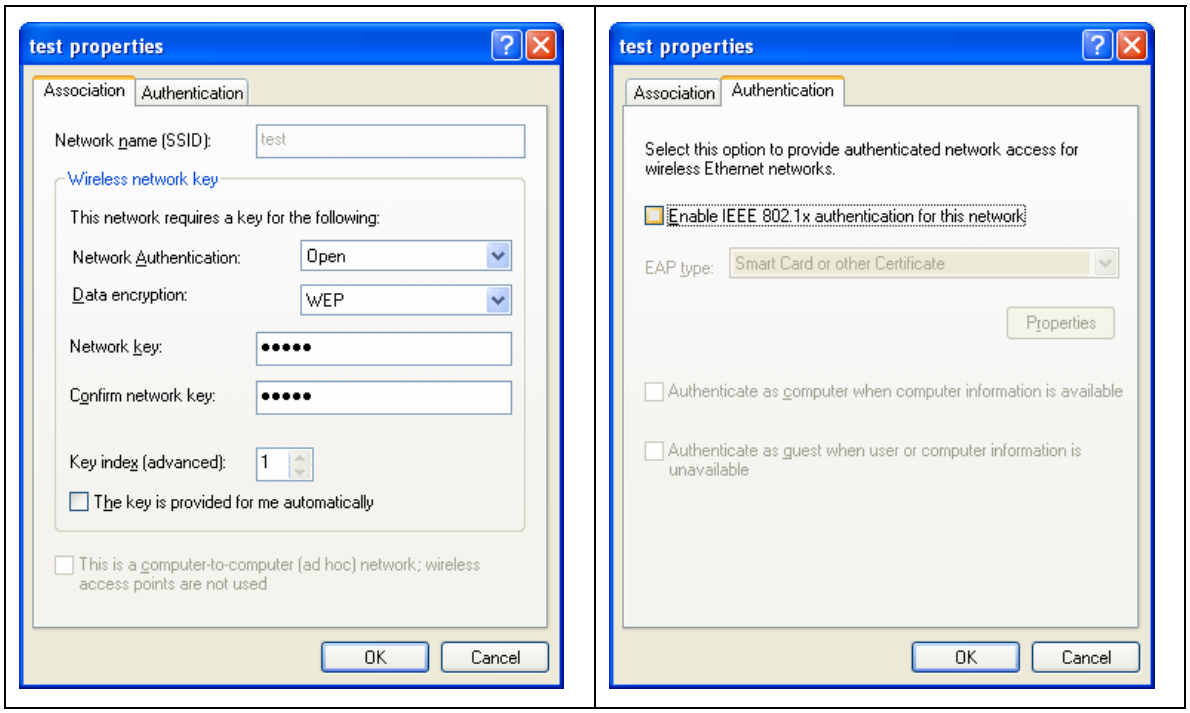


Figure 18. Windows XP Wireless Configuration with Static WEP Key.

3. With Dynamic WEP rekeying: The third option of the WEP encryption is using Dynamic WEP rekeying, in which the authentication server creates dynamic WEP keys that are changed in a predefined interval. The default changing interval for the HostAP is

300 seconds. This changing interval makes WEP cracking virtually impossible. Even if the network is congested and the WEP key is 64 bits, the allotted time is not sufficient for any cracking tool to break the WEP key.

The configuration of the XP client does not have any importance since the WEP key could not be set for the wireless connection. The attack only results in a denial-of-service attack against the legitimate client.

4. Using a Packet Generator

Packet generators allow the users to create any kind of packets and send them from one of the computer's network adapters. Excalibur is one of the most popular packer generators. For this thesis, Excalibur is used to generate an ICMP packet that is sent to a Web site (www.yahoo.com) to test the access to the network after the session is stolen. Excalibur provides options to input the MAC and IP addresses of the source and destination entities, as well as the other layer options of the packet (IP, TCP packet options.)

The MAC and IP address of the legitimate client, which are spoofed in the first step of the attack, are applied to the ICMP packet. One of the popular Web sites is selected as the destination of the ICMP ping packet. Figure 19 shows the interface of the Excalibur.

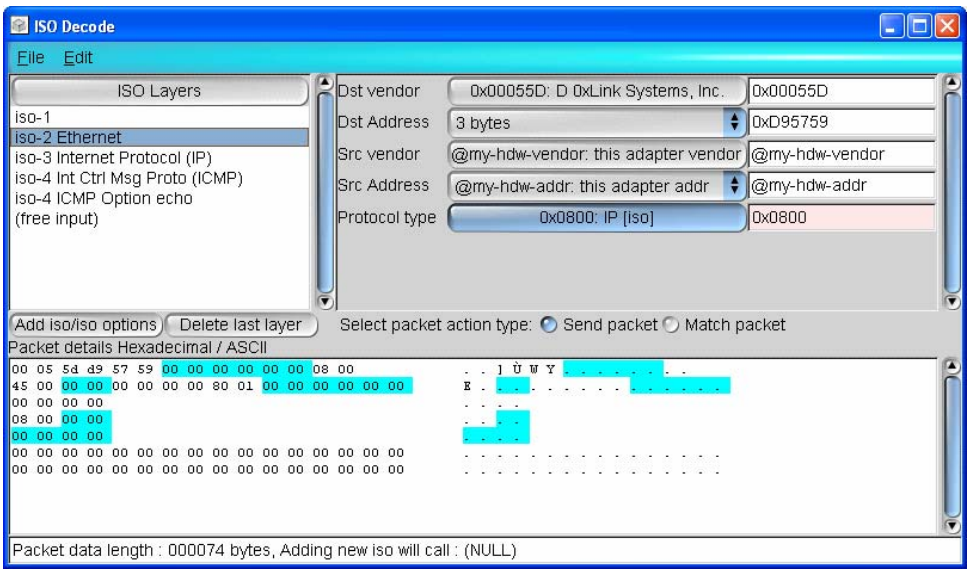


Figure 19. Excalibur Packet Generator

Since the XP clients cannot be configured to use the 802.1X authentication without using WEP encryption, the tests of the session hijacking attack without encryption could not be implemented. Since the legitimate user stays authenticated by the access point after it is disassociated from the network, the attacker can send the artificially generated packets by using the credentials of the legitimate client. For testing the packet generator, the access point is switched to open-system and WEP encryption is disabled. The ICMP ping packets are generated by the attacker and sent over the network in the name of the legitimate client. ICMP echo packets are received by the attacker via the access point. The network traffic can be received by the attacker after the authentication is bypassed.

When the WEP encryption is enabled, the ICMP packets cannot be sent by using the credentials of the attacker because there is no way of encrypting the generated ICMP ping packets with the WEP key before sending it. Excalibur does not have a feature that allows the generated packets to be encrypted with the WEP key. The packet generator could not be tested with the WEP encryption enabled.

D. RESULTS OF THE SESSION HIJACKING ATTACK

The University of Maryland paper [1] mentioned the success of the session hijacking attack without mentioning the status of the encryption. Both Cisco and Orinoco's responses both mentioned that the attack may be successful if encryption is not used.

After conducting the tests on the test-bed, the results were similar to the responses of Cisco and Orinoco. Without using the encryption, the attacker successfully hijacked the session. Even though the results were in favor of the successful attack, there were some points that prevented the conditions of the attack to exist. Commercial access points, like the one used in the tests (D-Link DWL 7000 AP), do not allow a network configuration that uses the 802.1X authentication without encryption. The software of the access points does not allow this kind of configuration. Additionally, the latest version of the Windows operating system (Windows XP SP1) does not have an option for making a connection that enables the 802.1X authentication without encryption. The warning

message of windows XP is seen in Figure 20. Since more than 90 percent of the computers use the Windows OS, the environment for using the 802.1X authentication seems impossible.

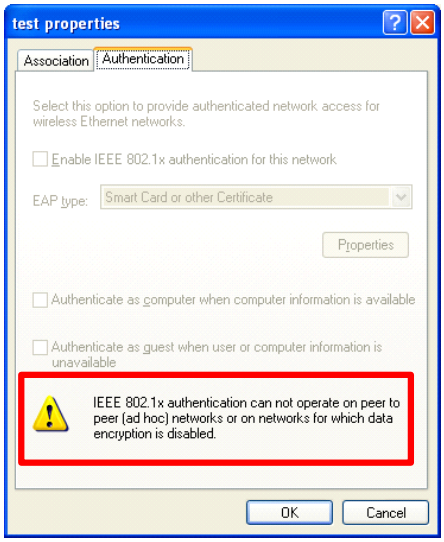


Figure 20. Windows XP Warning Message.

The tests with the use of static WEP keys showed that the attack would be possible but not efficient. The amount of time required to break the WEP key could be unworkable if the traffic of the victim network is not congested. If, however, the WEP key can be obtained then the other steps of the session hijacking attack can be performed. This possibility was tested artificially applying the WEP keys to the attacker client machine as if they were broken by the WEP crack tools. In theory, the attack is successful with the known encryption keys. Even if the network administrator changes the static WEP keys periodically the attacker should be able to crack the keys every time they are changed, given an insufficient change interval. The legitimate client will be kept out of the network with the disassociation packets sent from HostAP. The legitimate client cannot detect the existence of the attacker unless the administrator is notified about the problem. The administrator can detect the attacker that is using the credentials of the legitimate user that is unable to connect to the network.

The final test, with the Dynamic WEP keys, completely validated Cisco and Orinoco's responses. The rekeying period is small enough to prevent any kind of attempt to break the WEP key.

The 802.1X authentication is always vulnerable to injection of malicious frames. The disassociation of the legitimate client is always a possibility and can be achieved easily with the use of HostAP. However, the encryption is the greatest obstacle to overcome. If the encryption could not be beaten, then the attack becomes worthless. The packets sent by the malicious client are dropped since they are encrypted without the correct keys.

THIS PAGE INTENTIONALLY LEFT BLANK

V. RESULTS, 802.11I STANDAND AND SOLUTIONS

A. INTRODUCTION

The IEEE 802.11i group is working hard to find a complete solution to the type of session hijacking attacks demonstrated in Chapter IV. This chapter will examine that effort. Since the 802.11i working group has not finalized their work, this discussion is based on the most recent draft recommendations published by the group regarding the 802.11 and 802.1X standards. (http://grouper.ieee.org/groups/802/11/LetterBallots.html) The draft recommendations are discussed generally and the sections that are concerned with the session hijacking attack are specifically analyzed.

Our analysis has uncovered some inadequacies in the 802.11i recommendations with regard to their effectiveness against session hijacking attacks. Possible solutions to the inadequacies are proposed at the end of the chapter.

B. PROBLEMS OF 802.1X AUTHENTICATION STANDARD IN WIRELESS NETWORKS

While describing the session hijacking attack in Chapter II, the vulnerabilities of the 802.1X authentication standard that allows the attackers to launch the session hijacking attack were discussed. The problems can be summarized as follows:

1. The client maintains a state machine and the authenticator maintains a state machine for each client. These two state machines may be out of synchronization as described in Chapter II, which could be exploited for a session hijacking attack.

2. The lack of authenticity of the management frames of the authentication traffic.

3. One-way authentication of the client only.

4. Inadequacy of WEP encryption.

5. Open-signal propagation medium

C. 802.11i STANDARD

In this chapter, the new wireless security standard will be covered. The new standard is called 802.11i. The final version of the standard is not completed, it is still a work in progress; however, the latest draft of the standard was published after it passed the ballot of the wireless committee. 802.11i draft 7 is the latest draft of the standard.

1. History

In July of 1999, the IEEE 802.11 had a study group meeting for people interested in enhancing the IEEE 802.11 MAC for better quality of service (QoS) and privacy. In March of 2000, TGe was created to enhance the 802.11 Medium Access Control (MAC) to improve and manage QoS, and to provide enhanced security and authentication mechanisms for wireless networks. In March of 2001, the TGe was split into separate working groups: TGe and TGi to focus on QoS and security issues, respectively. TGi began acting independently in May of 2001. Since then, TGi has worked on the new standard and published the drafts of this new standard after the Letter Ballots. Draft 7 is the latest draft of the 802.11i standard.

2. Architecture of 802.11i

In general, the new 802.11i standard consists of two layers and there are three main pieces that are organized in these two layers. On the lower level, there are two improved encryption algorithms: the Temporal Key Integrity Protocol (TKIP) and the Counter mode with CBC-MAC protocol (CCMP). Both encryption protocols are designed to protect the data integrity better than the legacy WEP encryption technique. TKIP is targeted at legacy equipment and CCMP is targeted at future WLAN equipment. On the upper level, there is the authentication standard 802.1X. 802.1X was not a part of the original 802.11 standard and it becomes a part of the wireless LAN standards with 802.11i.

The goal of 802.11i is to combine the newly defined standards, such as TKIP and CCMP, with the existing 802.11 standards and 802.1X authentication standard. In order to understand how these three pieces fit together, the details of each individual standard

should be examined. Since the 802.1X standard has been examined in detail in the previous chapters, it is omitted from this discussion.

3. Key Management

802.1X authentication is carried over to the new standard, however, key management issues have been modified to make it more robust. The new standard defines two key hierarchies: pairwise key hierarchy, which aims to protect unicast traffic, and group-key hierarchy to protect multicast traffic.

a. Pairwise Key Hierarchy

The new standard brings the use of Pairwise Master Key (PMK) for the security of the session between the supplicant and the authenticator. A Pairwise Transient Key (PTK) is derived from the PMK, which will be chopped into three different keys to be used in different fields of the security. These three keys are: the EAPOL-Key MIC Key (KCK), the EAPOL-Key Encryption Key (KEK), and the Temporal Key (TK).

Depending on the EAP type of authentication, the supplicant and the authentication server share some sort of key material before the actual authentication starts. Although it is not part of the new standard, certificate-based EAP-TLS is the most popular and convenient EAP type. Certificates are installed into both the supplicant and the authentication server before the actual authentication takes place. Master Key is derived during the 802.1X EAP Authentication. Only the Authentication Server (AS) and the Supplicant can have this Master Key.

PMK can be defined as a fresh symmetric key, which controls AS's and Supplicant's access to the channel during the session. PMK results from the authentication between the supplicant and the AS. PMK generation is normally done independently and simultaneously on the AS and the supplicant, based on the information communicated between the two entities during the authentication. Each EAP method can derive the PMK from the Master Key in different ways. The PMK is pushed to the AP afterwards. Figure 21 shows the Pairwise Key Hierarchy.

PRF is a Pseudo-Random Function that is used in a number of places in the new standard. PRF generates different length outputs that vary from 128 bit to 512 bits. The values inside the brackets in Figure 21 are the inputs of PRF.

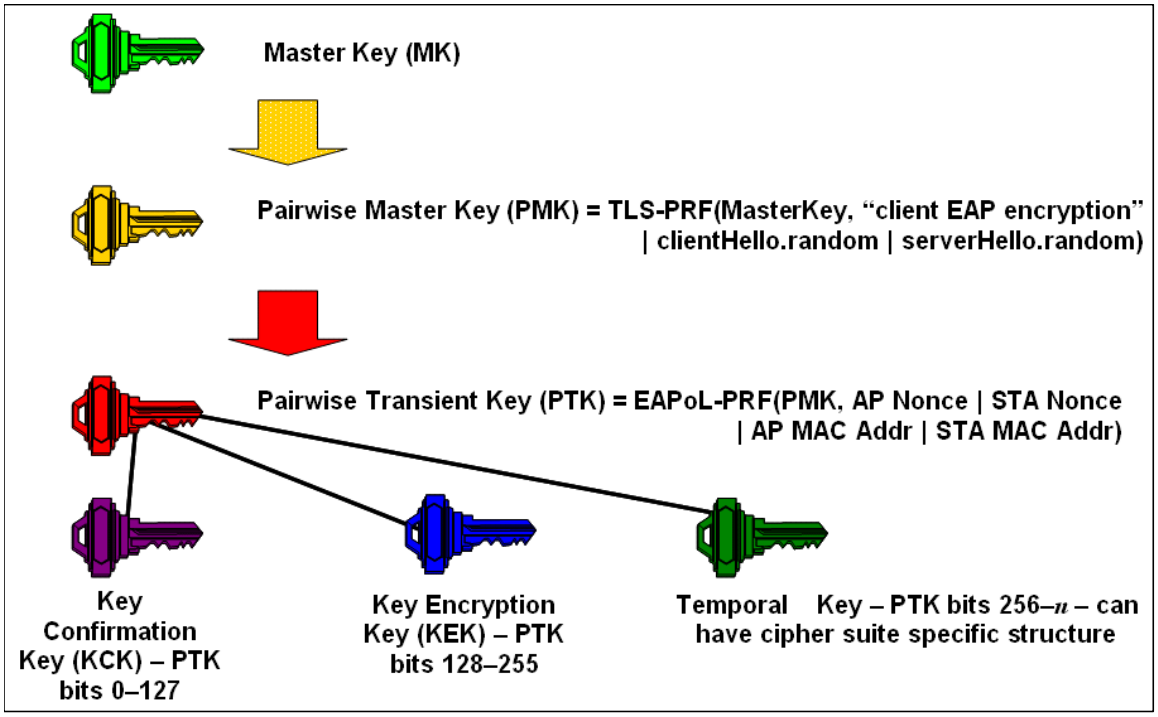


Figure 21. Pairwise Key Hierarchy (From Ref 22)

The Pairwise Transient Key (PTK) is derived from the Pairwise Master Key by using the authenticator's and the supplicant's MAC addresses along with the AP nonce and Supplicant Nonce values. Both nonce values are random or pseudo-random values contributed to by their respective entities. The length of PTK is 512 bits for TKIP and 384 for CCMP, WRAP and WEP. TKIP, WRAP and CCMP are new encryption standards, which will be discussed later in this chapter.

The Key Confirmation Key (KCK) is computed as the first 128 bits of the PTK and is used by the 802.1X to provide data origin authenticity in the 4-way handshake and group-key distribution messages. The Key Encryption Key (KEK) is computed as the second 128 bits of the PTK. The KEK is used to provide confidentiality in the 4-wap handshake and group-key distribution messages. The Temporal Key (TK) is the remaining part of the PTK, in 128 bits chunks. If the remaining part of the PTK is

more than 128 bits long, more than one TK can be produced. The TK is used to secure the data traffic.

b. Group Key Hierarchy

Similar to the Pairwise Key Hierarchy, a Group Transient Key (GTK) is generated from the Group Master Key (GMK). This GTK is partitioned into temporal keys used to protect the broadcast and multicast communication. These group keys are used between a single authenticator and all the supplicants authenticated to that authenticator.

c. Four-Way Handshake

Before the 802.1X authentication, the supplicant and the authenticator exchange discovery packets. In the discovery period, the authenticator advertises its capabilities and the Supplicant sends back its association request based on the authenticator's capabilities. After the discovery, both stations are ready to authenticate over the established 802.11 channel.

After the 802.1X authentication, the participating entities advance to the key management step. In this step, the Pairwise Master Key (PMK) is generated both in the authentication server and the supplicant. The PMK is then pushed to the authenticator by the AS. A Four-Way Handshake process takes place between the supplicant and the authenticator to produce and derive the PTK and other keys. Figure 22 shows the four-way handshake process.

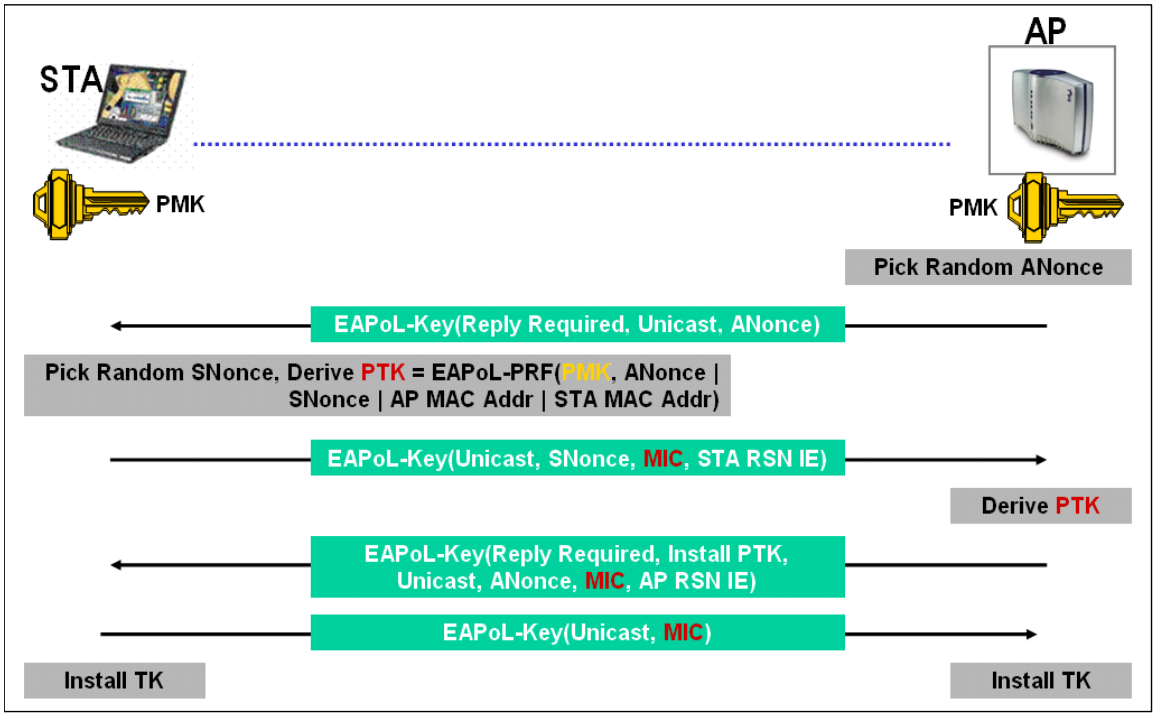


Figure 22. Four-Way Handshake (From Ref. 22)

The authenticator picks a nonce which is a random, one-time- use value (ANonce) and sends it to the supplicant within an EAPOL-Key Packet as the first message. On receiving the first message, the supplicant generates its own nonce (SNonce) value and derives the PTK from the PMK, nonce values, and the MAC addresses. After derivation of the PTK, the supplicant sends the second message to the authenticator. The authenticator receives the nonce value of the supplicant and generates the exact same PTK value on the authenticator side. The authenticator sends another message back to the supplicant indicating that it possesses the PTK. The last packet is send from the supplicant to the authenticator, which ends the 4-way handshake.

The four-way handshake establishes a fresh Pairwise Key that is bound to the supplicant and the access point for the session. It also provides a security check that there is no man-in-the-middle between the two PTK holder entities.

4. Temporal Key Integrity Protocol (TKIP) Overview

The TKIP was designed to address all the known attacks against the WEP algorithm while still maintaining backward compatibility with the legacy hardware. It was designed to be made available as a firmware or software upgrade to existing hardware so that users would be able to upgrade their level of security without replacing existing equipment or purchasing new hardware. The TKIP provides an upgrade path by offering an additional protocol or a wrapper around the WEP. The TKIP is comprised of the following elements:

1. A message integrity code (MIC) provides a keyed cryptographic checksum using the source and destination MAC addresses and the plaintext data of the 802.11 frames, which protects the session against forgery attacks.

2. Countermeasures are implemented by the TKIP to limit the probability of successful forgery and the amount of information that an attacker can learn about a particular key.

3. A TKIP sequence counter with a 48-bit Initialization Vector (IV) is also implemented to sequence the packets that are sent. This provides a replay protection, which is not sufficient enough to protect against these kinds of attacks. Any fragmented packets received out of order are dropped by the receiver.

4. Per packet key mixing of the IV is used to break up the correlation used by weak key attacks.

Figure 23 shows the picture of a TKIP encrypted MAC protocol data unit (MPDU). Different from the original WEP protocol, TKIP introduces the use of an extended 48-bit IV, which is called the TKIP sequence counter (TSC). The TSC is updated in each packet, which aims to extend the life of the temporal key and eliminate the need to re-key the temporal key in a single association. Using 48 bits, 2^{48} packets can be exchanged using the same temporal key. It will take about 100 years for a key reuse to occur under normal network traffic conditions.

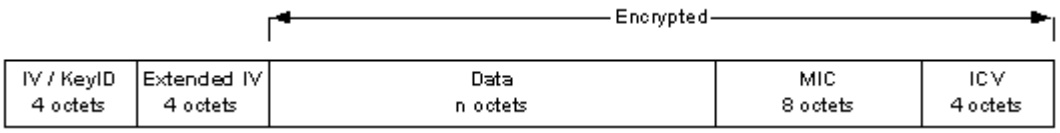


Figure 23. MPDU Format after TKIP Encryption (From Ref. 27)

To construct the TSC for an individual packet, the first and second bytes from the original WEP IV and the four bytes provided in the extended IV are used. The TKIP encapsulation process is shown in Figure 24. The temporal and MIC keys are used, which are derived from the PMK generated as part of the 802.1X exchange. [27]

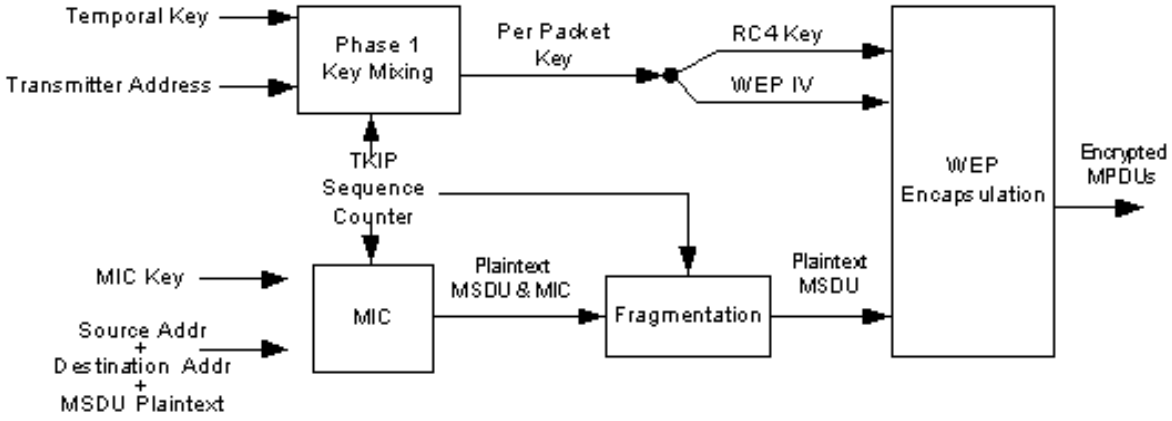


Figure 24. Diagram Depicting the TKIP Encapsulation Process (From Ref. 27)

The per-packet key in TKIP is 128 bits and is comprised of a 104-bit long RC4 key and 24-bit IV. The key is produced after the mixture of the Temporal Key, the Transmitter Address and the TSC in a two-phase key mixing function. [27]

The WEP integrity check Vector (ICV) is computed by a simple CRC-32 cyclic redundancy check. The TKIP replaces CRC-32 with MIC (nicknamed as Michael), which is a stronger, one-way hash function. The MIC value is calculated using the TSC, source and destination MAC addresses, Plaintext and the MIC key. After computing the MIC, the packet is keyed to the sender and receiver, preventing attacks based on packet

forgery. The MIC makes it much more difficult for an attacker to intercept and alter the packets in favor of their attacks. [27]

The decapsulation process is essentially the same as the process illustrated in Figure 14 with the following exceptions. The countermeasures that are mentioned before are applied to the packet. The TSC value of the packet is extracted, and it is compared to the previously received packets. If the TSC value of the recent packet is smaller than the previously received packets, the new packet is discarded in order to prevent potential replay attacks.

On the receiving side, the MIC value is calculated and compared to the received MIC value. If the MIC values do not match, then the countermeasures are invoked. The countermeasures are a warning sent to the administrator of a possible attack and rekeying the temporal key.

5. The Counter-Mode/CBC-MAC Protocol (CCMP) Overview

The new encryption method defined in 802.11i is based on a modern encryption technique called the Advanced Encryption Standard (AES). The AES can be used in a variety of different modes and algorithms. Counter mode with CBC-MAC (CCM) is chosen for the new standard in which the counter mode AES provides the data privacy and the Cryptographic Block Cipher–Message Authentication Code (CBC-MAC) delivers data integrity and authentication. [27]

AES is a modern encryption algorithm that is a symmetric iterated block cipher. The same key is used for both encryption and decryption. The AES standard uses 128-bit blocks for encryption, and for 802.11 the encryption key length is also fixed at 128 bits.

Figure 25 shows the format of an AES encrypted MAC-layer Protocol Data Unit (MPDU). The packet is expanded by 16 bytes over an unencrypted frame. [27]

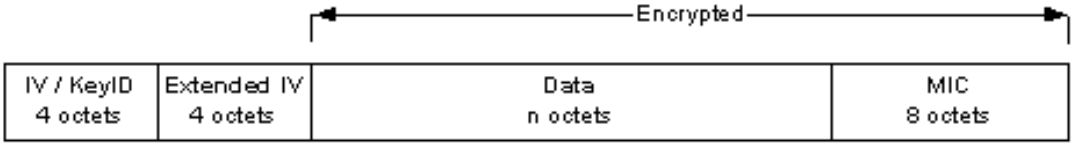


Figure 25. MPDU Format after CCMP Encryption (From Ref. 27)

The CCMP also uses a 48-bit IV called a Packet Number (PN) which is used to create the AES cipher for both the MIC and frame encryption. Figure X shows the CCMP encapsulation process.

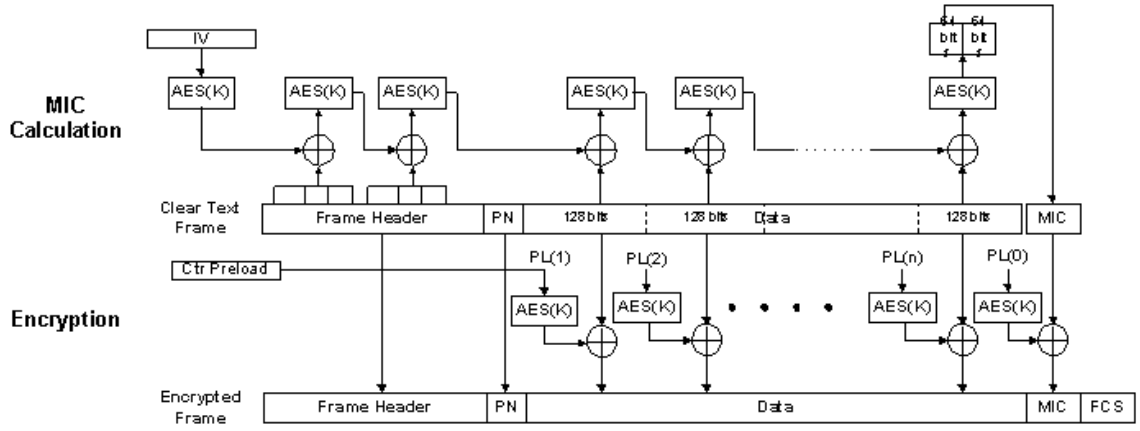


Figure 26. Diagram of the CCMP Encapsulation Process (From Ref. 27)

The CCMP encapsulation process has two parts: Encryption and the MIC calculation. A temporal encryption key is used for both the MIC calculation and the packet encryption. The Temporal Key is derived form the master key, which is set to both entities after the 802.1X encryption. [27]

Figure 26 shows the encryption and MIC calculation diagram of CCMP. Since AES is a block-cipher algorithm, the overall process is more complicated than both TKIP and WEP. The IV is the seed of the MIC calculation with the PN and some specific data that comes from the header of the particular frame. The IV first goes to the AES ciphering block and the output is XORed with some other elements of the frame header. The result is an input to the next AES block. This ciphering process continues until the end of the frame. At the end, a 128-bit CBC-MAC value is computed. The upper 64-bit part of this MAC is taken out to be used in the final MIC which is appended to the encrypted frame. [27]

The encryption process, similar to the MAC calculation process, is seeded by a counter preload value, which is formed from the PN. In addition to the counter preload value, data from the frame header is also used for the encryption seed. The counter value is initialized to 1 at the start. The clear text is chopped into 128-bit chunks which are

XORed with the output of the previous AES block. The counter value is incremented by one and the same process continues on until the end of the clear text. The last and final counter value is set to 0 and its output from the AES block is XORed with the MIC value which is computed in the previous MIC Calculation step. The output of this XOR operation is appended to the end of the encrypted frame.

The decapsulation of the encapsulated CCMP frame is not much different than the encapsulation process. It is essentially the reverse of the encapsulation shown in Figure 16. The decapsulation process has one more step of comparing the MIC values of the received frame with the computed MIC value of the same frame.

6. Implementation of 802.11i

The Wi-Fi Alliance is a nonprofit international association formed in 1999 to certify interoperability of wireless Local Area Network products based on the IEEE 802.11 specification. In 2001, there were 100 Wi-Fi Certified Products and today there are more than 500 such products. The rapid growing wireless industry is demanding a more secure wireless environment and cannot wait for the 802.11i standard to be ratified, probably in 2004. Based on the immediate need for security, the goal of this alliance is to implement what is stable in 802.11i and bring it to market in Wi-Fi Protected Access (WPA).

a. Wi-Fi Protected Access (WPA) Overview

The first goal of the Wi-Fi alliance is to solve the security problems of the wireless networks with firmware and software upgrades without changing the hardware. WPA is a subset of the existing 802.11i draft, which uses TKIP instead of WEP. The WPA standard might not be fully compatible with some legacy devices and operating systems; however, it is designed to be forward compatible with the 802.11i standard.

b. Robust Security Network (RSN) Overview

The RSN is the enhanced security standard aimed at addressing the vulnerabilities of 802.1X and encryption issues concerning TKIP and CCMP. Simultaneous use of TKIP and CCMP is supported by RSN, however only CCMP will be mandatory while keeping the TKIP optional for true RSN implementations.

The authentication standard chosen for RSN is 802.1X and EAP, while the encryption algorithm will be the AES. Besides the new and strong functionality of RSN, it will run very poorly on some legacy devices because the AES algorithm does not perform well on the legacy devices. Table 1 presents some important parameters of the security standards.

		WPA	802.11i
	WEP	TKIP	AES-CCMP
Cipher	RC4	RC4	AES
Key Size	40 or 104 bits	128 bits encryption, 64 bit auth	128 bits
Key Life	24-bit IV, wrap	48-bit IV	48-bit IV
Packet Key	Concat.	Mixing Fnc	Not Needed
Integrity			
Data	CRC-32	Michael	CCM
Header	None	Michael	CCM
Replay	None	Use IV	Use IV
Key Mgmt.	None	EAP-based	EAP-based

Table 3. Comparison of the Existing and Emerging Security Standards

D. SOLUTIONS AND DISCUSSION

The first part of this chapter discussed the problems of the wireless networks while the second part of the chapter discussed the new wireless security standard (802.11i). In this conclusion, the problems and the solutions will be discussed together.

Since TGi is going to include the 802.1X standard as the authentication method in the 802.11i security standard, the advantages and disadvantages of the 802.1X standard will be carried over to the new standard.

1. Mutual Authentication

The lack of mutual authentication was one of the most important security vulnerabilities of wireless networks. The 802.1X standard will be used as the authentication method; however, some vulnerable and weak points are rectified. EAP is used to carry the authentication traffic. The new standard mandates the use of EAP

methods, which provide mutual authentication. EAP-TLS is the most popular and widely supported method that provides mutual authentication. Windows XP contains an implementation of EAP-TLS. Thus, EAP-TLS becomes the de facto authentication method. By using EAP-TLS and discarding the other weak methods, mutual authentication can be accomplished.

The authentication problem can only be solved by employing mutual authentication. EAP-TLS should be implemented by the networks that need security.

2. Encryption and Key Management

The key management part of the 802.1X standard is modified in the new standard to provide a better way of encrypting the network traffic. Key management is the most important improvement that is introduced by the new standard. Both of the new techniques, TKIP and CCMP, provide better and improved encryption than WEP. Besides the encryption of the traffic, per-packet authentication and message integrity checking is improved.

Since TKIP improves the WEP keys with enhancements such as 48-bit IV, the possibility of breaking the WEP key is quite difficult for an attacker. CCMP and WRAP both are based on a better encryption algorithm: AES. The complicated nature of this algorithm will provide a robust encryption for the wireless networks.

The proposed four-way handshake mechanism establishes a fresh set of pairwise keys for each new session between a supplicant and the authenticator. The mechanism is carefully designed to eliminate the possibilities of a man-in-the-middle attack between the two entities during the handshake period. [22]

TKIP is applicable to the legacy devices, while CCMP will run only on the new devices. Even if neither of these two techniques is implemented, some security precautions can still be applied to harden the wireless networks: hiding the SSID, changing the static WEP key frequently, applying WEP re-keying (if applicable).

3. Management Frames Authentication

The session hijacking attack, which is defined in the previous chapter, starts with the disassociation of the supplicant with a fake disassociation frame. The cause of this vulnerability is the unauthenticated management frames. The draft of the new security standard keeps the open-system authentication prior to 802.1X authentication. The open-system association is used to exchange the capabilities of the entities. Although the open-system authentication does not serve any security role, the possibility of sending disassociation frame still exists. To mitigate this vulnerability, the management frames should also be authenticated.

The 802.1X authentication protocol is used in the new standard with modifications to the key management issues. All the other parts of the standard remain unchanged. With the use of a new and complicated key management scheme, the attackers should be unable to use any network sources without breaking the key management.

The supplicant can still be disconnected from the network, but the attacker will not have any chance of using the network. The network traffic will not be unencrypted or weakly-encrypted; instead the encryption will be strong enough to limit the attempt to hijack the session to a denial of service attack, at worst.

E. SUMMARY

By using the new security standard, the necessary conditions of the session hijacking attack mentioned in Chapter IV are eliminated. The first two necessary conditions are precluded by the use of the EAP methods that provide mutual authentication. The other important issue is the new key management, which also helps prevent the first two necessary conditions from happening.

The fourth necessary condition will be prevented by the use of new encryption techniques, which are much stronger than WEP. The new encryption techniques, combined with the new-key management issues, dismiss the possibility of breaking the encryption.

The possibility of sending a disassociation frame still exists. The open-system association is the part of the new standard used to exchange the parameters of the connection. With the new and robust encryption techniques, this vulnerability can only end up as a denial-of-service attack against the client at worst.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION AND FUTURE WORK

A. CONCLUSION

This thesis examined the 802.1X authentication standard. The standard is explained in detail, and the entities and the protocols of the standard are covered in depth. The security vulnerabilities of the standard are discussed.

The security vulnerabilities of the 802.11 WLAN standard will be addressed when the new security standard (802.11i) is completed and published by the security working group (TGi). TGi considers the 802.1X standard a very important and key part of its solution. The new standard will introduce 802.1X as a solution to the authentication problems of the 802.11 standard.

In this thesis, the security vulnerabilities of 802.1X are tested. The vulnerabilities were discovered and published in a University of Maryland's paper. [1] For testing purposes, an open-source test-bed was built. In the test-bed, the EAP-TLS authentication method is chosen and implemented. For this method, public key certificates are created and used. The test-bed was used to implement a session hijacking attack that is mentioned in the University of Maryland paper. [1] The attack is divided into three main parts and each part is performed individually.

The experiments showed that launching session-hijacking attacks against wireless networks is easier than the wired networks because of the open broadcast nature of the wireless networks. Another important outcome of the experiments is the importance of encryption. Encryption and key management are the most important components of a secure network. If the encryption is strong and key management is well-organized, the attackers cannot reach their goals even if they exploit some other vulnerabilities. The new encryption methods that are proposed by the new wireless security standard (802.11i) provide much better security than the WEP encryption.

The authentication and access control issues are addressed by the use of 802.1X authentication standard. The result of the experiment demonstrated that this standard provides mutual authentication if the authentication method is chosen right. EAP-TLS is a certificate-based authentication method that provides mutual authentication.

Open system authentication is proposed by the IEEE security working group (TGi) for the exchange of authentication information. The use of an open-system authentication still allows the attackers to disassociate a legitimate client. This vulnerability, however, is only enough for a denial-of-service attack, not a session hijacking attack.

B. FUTURE WORK

In this thesis a session hijacking attack is evaluated with an open-source test-bed. Since the test-bed is built and serves as a good environment for testing, other types of attack can be evaluated and other security problems can be discovered. Other authentication methods, such as EAP-MD5, can be tested using the test-bed.

The new authentication scheme proposes new encryption techniques such as CCMP, WRAP and TKIP, as well as a new key management scheme. Once the new 802.11i standard is completed, the new standard, along with the authentication methods, key management scheme, and encryption techniques, can be evaluated and tested.

The source code of the HostAP and Linux client code Xsupplicant should be manipulated to employ the tests. Since the test-bed is built on Linux OS and the source code is written in C language, Experience with C and C++ programming and Linux OS is mandatory for this type testing.

APPENDIX A

A. CERTIFICATE GENERATOR CONFIGURATION

1. OpenSSL Configuration File

```
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#

# This definition stops the following lines choking if HOME isn't
# defined.
HOME            = .
RANDFILE        = $ENV::HOME/.rnd

# Extra OBJECT IDENTIFIER info:
#oid_file       = $ENV::HOME/.oid
oid_section     = new_oids

# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions            =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)

[ new_oids ]

# We can add new OIDs in here for use by 'ca' and 'req'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6

####################################################################
[ ca ]
default_ca  = CA_default            # The default ca section

####################################################################
[ CA_default ]

dir         = ./demoCA        # Where everything is kept
certs       = $dir/certs            # Where the issued certs are kept
crl_dir         = $dir/crl          # Where the issued crl are kept
database    = $dir/index.txt  # database index file.
new_certs_dir       = $dir/newcerts         # default place for new
certs.

certificate = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
crl         = $dir/crl.pem          # The current CRL
private_key = $dir/private/cakey.pem # The private key
RANDFILE    = $dir/private/.rand    # private random number file
```

```
x509_extensions   = usr_cert        # The extentions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
# name_opt  = ca_default            # Subject Name options
# cert_opt  = ca_default            # Certificate field options

# Extension copying option: use with caution.
# copy_extensions = copy

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2
CRLs
# so this is commented out by default to leave a V1 CRL.
# crl_extensions  = crl_ext

default_days      = 365             # how long to certify for
default_crl_days= 30                # how long before next CRL
default_md  = md5             # which md to use.
preserve    = no              # keep passed DN ordering

# A few difference way of specifying how similar the request should
look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy            = policy_match

# For the CA policy
[ policy_match ]
countryName       = match
stateOrProvinceName     = match
organizationName  = match
organizationalUnitName  = optional
commonName        = supplied
emailAddress            = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName       = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName  = optional
organizationalUnitName  = optional
commonName        = supplied
emailAddress            = optional

####################################################################
[ req ]
default_bits            = 1024
default_keyfile   = privkey.pem
distinguished_name      = req_distinguished_name
attributes        = req_attributes
x509_extensions   = v3_ca     # The extentions to add to the self
signed cert

# Passwords for private keys if not present they will be prompted for
```

```
# input_password = secret
# output_password = secret

# This sets a mask for permitted string types. There are several
options.
# default: PrintableString, T61String, BMPString.
# pkix       : PrintableString, BMPString.
# utf8only: only UTF8Strings.
# nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
# MASK:XXXX a literal mask value.
# WARNING: current versions of Netscape crash on BMPStrings or
UTF8Strings
# so use this option with caution!
string_mask = nombstr

# req_extensions = v3_req # The extensions to add to a certificate
request

[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
countryName_default             = US
countryName_min                 = 2
countryName_max                 = 2

stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = California

localityName                    = Locality Name (eg, city)
localityName_default            = Monterey

0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = NPGS

# we can do this but it is not needed normally :-)
#1.organizationName             = Second Organization Name (eg, company)
#1.organizationName_default     = World Wide Web Pty Ltd

organizationalUnitName          = Organizational Unit Name (eg, section)
organizationalUnitName_default      = SAAM

commonName                      = Common Name (eg, YOUR name)
commonName_max                  = 64
commanName_default              = WirelessSAAM CA

emailAddress                    = Email Address
emailAddress_max                = 64
emailAddress_default            = oozan@nps.navy.mil

# SET-ex3                       = SET extension number 3

[ req_attributes ]
challengePassword               = A challenge password
challengePassword_min           = 4
challengePassword_max           = 20

unstructuredName        = An optional company name
```

```
[ usr_cert ]

# These extensions are added when 'ca' signs a request.

# This goes against PKIX guidelines but some CAs do it and some
software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType                    = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment               = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl              = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

[ v3_req ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
```

```
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v3_ca ]

# Extensions for a typical CA

# PKIX recommendation.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer:always

# This is what PKIX recommends but some broken software chokes on
critical
# extensions.
#basicConstraints = critical,CA:true
# So we do this instead.
basicConstraints = CA:true

# Key usage: this is typical for a CA certificate. However since it
will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
# keyUsage = cRLSign, keyCertSign

# Some might want this also
# nsCertType = sslCA, emailCA

# Include email address in subject alt name: another PKIX
recommendation
# subjectAltName=email:copy
# Copy issuer details
# issuerAltName=issuer:copy

# DER hex encoding of an extension: beware experts only!
# obj=DER:02:03
# Where 'obj' is a standard or added object
# You can even override a supported extension:
# basicConstraints= critical, DER:30:03:01:01:FF

[ crl_ext ]

# CRL extensions.
# Only issuerAltName and authorityKeyIdentifier make any sense in a
CRL.

# issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always,issuer:always
```

B. CERTIFICATE GENERATION SCRIPTS

1. Root Certificate Authority Generation Script

```
#!/bin/sh
SSL=/usr/local/openssl-certgen
export PATH=${SSL}/bin/:${SSL}/ssl/misc:${PATH}
export LD_LIBRARY_PATH=${SSL}/lib
# needed if you need to start from scratch otherwise the CA.pl -newca
command doesn't copy the new
# private key into the CA directories
rm -rf demoCA
echo
"**********************************************************************
***********"
echo "Creating self-signed private key and certificate"
echo "When prompted override the default value for the Common Name
field"
echo
"**********************************************************************
***********"
echo
# Generate a new self-signed certificate.
# After invocation, newreq.pem will contain a private key and
certificate
# newreq.pem will be used in the next step
openssl req -new -x509 -keyout newreq.pem -out newreq.pem -passin
pass:whatever -passout pass:whatever
echo
"**********************************************************************
***********"
echo "Creating a new CA hierarchy (used later by the "ca" command) with
the certificate"
echo "and private key created in the last step"
echo
"**********************************************************************
***********"
echo
echo "newreq.pem" | CA.pl -newca >/dev/null
echo
"**********************************************************************
***********"
echo "Creating ROOT CA"
echo
"**********************************************************************
***********"
echo
# Create a PKCS#12 file, using the previously created CA
certificate/key
# The certificate in demoCA/cacert.pem is the same as in newreq.pem.
Instead of
# using "-in demoCA/cacert.pem" we could have used "-in newreq.pem" and
then omitted
# the "-inkey newreq.pem" because newreq.pem contains both the private
key and certificate
```

```
openssl pkcs12 -export -in demoCA/cacert.pem -inkey newreq.pem -out
root.p12 -cacerts -passin pass:whatever -passout pass:whatever
# parse the PKCS#12 file just created and produce a PEM format
certificate and key in root.pem
openssl pkcs12 -in root.p12 -out root.pem -passin pass:whatever -
passout pass:whatever
# Convert root certificate from PEM format to DER format
openssl x509 -inform PEM -outform DER -in root.pem -out root.der
#Clean Up
rm -rf newreq.pem
```

2. Server Certificate Generation Script

```
#!/bin/sh
SSL=/usr/local/openssl-certgen
export PATH=${SSL}/bin/:${SSL}/ssl/misc:${PATH}
export LD_LIBRARY_PATH=${SSL}/lib
echo
"**********************************************************************
***********"
echo "Creating server private key and certificate"
echo "When prompted enter the server name in the Common Name field."
echo
"**********************************************************************
***********"
echo
# Request a new PKCS#10 certificate.
# First, newreq.pem will be overwritten with the new certificate
request
openssl req -new -keyout newreq.pem -out newreq.pem -passin
pass:whatever -passout pass:whatever
# Sign the certificate request. The policy is defined in the
openssl.cnf file.
# The request generated in the previous step is specified with the -
infiles option and
# the output is in newcert.pem
# The -extensions option is necessary to add the OID for the extended
key for server authentication
openssl ca -policy policy_anything -out newcert.pem -passin
pass:whatever -key whatever -extensions xpserver_ext -extfile
xpextensions -infiles newreq.pem
# Create a PKCS#12 file from the new certificate and its private key
found in newreq.pem
# and place in file specified on the command line
openssl pkcs12 -export -in newcert.pem -inkey newreq.pem -out $1.p12 -
clcerts -passin pass:whatever -passout pass:whatever
# parse the PKCS#12 file just created and produce a PEM format
certificate and key in certsrv.pem
openssl pkcs12 -in $1.p12 -out $1.pem -passin pass:whatever -passout
pass:whatever
# Convert certificate from PEM format to DER format
openssl x509 -inform PEM -outform DER -in $1.pem -out $1.der
# Clean Up
rm -rf newert.pem newreq.pem
```

3. Supplicant Certificate Generation Script

```
#!/bin/sh
SSL=/usr/local/openssl-certgen
export PATH=${SSL}/bin/:${SSL}/ssl/misc:${PATH}
export LD_LIBRARY_PATH=${SSL}/lib
echo
"**********************************************************************
***********"
echo "Creating client private key and certificate"
echo "When prompted enter the client name in the Common Name field.
This is the same"
echo " used as the Username in FreeRADIUS"
echo
"**********************************************************************
***********"
echo
# Request a new PKCS#10 certificate.
# First, newreq.pem will be overwritten with the new certificate
request
openssl req -new -keyout newreq.pem -out newreq.pem -passin
pass:whatever -passout pass:whatever
# Sign the certificate request. The policy is defined in the
openssl.cnf file.
# The request generated in the previous step is specified with the -
infiles option and
# the output is in newcert.pem
# The -extensions option is necessary to add the OID for the extended
key for client authentication
openssl ca -policy policy_anything -out newcert.pem -passin
pass:whatever -key whatever -extensions xpclient_ext -extfile
xpextensions -infiles newreq.pem
# Create a PKCS#12 file from the new certificate and its private key
found in newreq.pem
# and place in file specified on the command line
openssl pkcs12 -export -in newcert.pem -inkey newreq.pem -out $1.p12 -
clcerts -passin pass:whatever -passout pass:whatever
# parse the PKCS#12 file just created and produce a PEM format
certificate and key in certclt.pem
openssl pkcs12 -in $1.p12 -out $1.pem -passin pass:whatever -passout
pass:whatever
# Convert certificate from PEM format to DER format
openssl x509 -inform PEM -outform DER -in $1.pem -out $1.der
# clean up
rm -rf newcert newreq.pem
```

4. XP Specific Extension Files

```
[xpclient_ext]
extendedKeyUsage = 1.3.6.1.5.5.7.3.2
[xpserver_ext]
extendedKeyUsage = 1.3.6.1.5.5.7.3.1
```

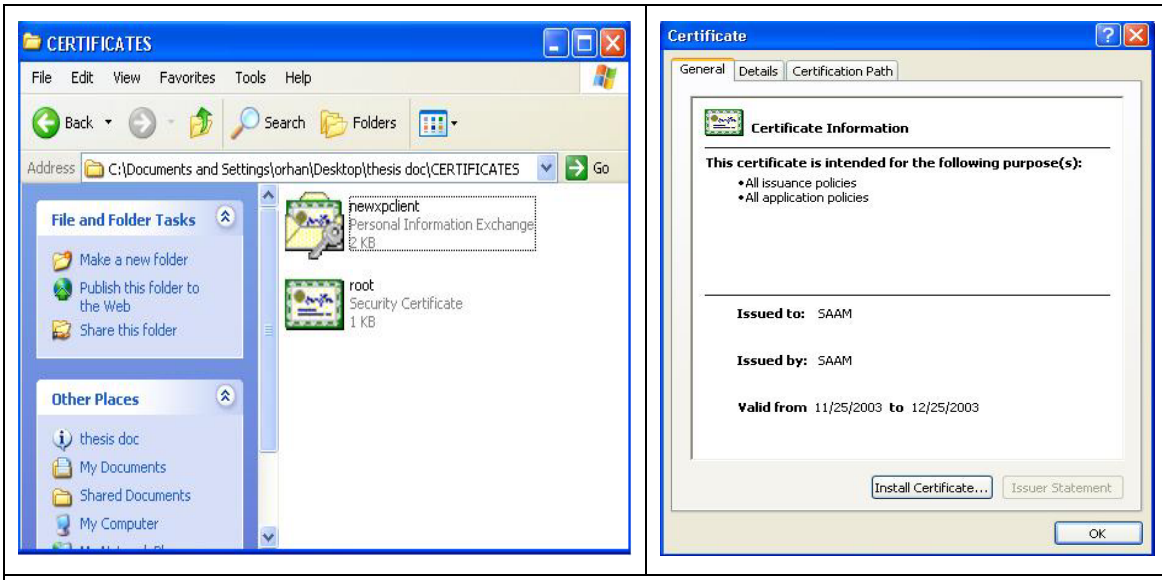
APPENDIX B

A. WINDOWS XP CERTIFICATE INSTALLATION

Windows XP with Service Pack 1 support requires a client public key certificate, a private key corresponding to this public key and a rootCA public key certificate to verify the server's certificate authentication.

The "newxpclient.p12" file contains the public key certificate and private key of the client. The "root.der" contains the public key certificate (PKC) of the root certificate authority. We must assure that both of these files are generated in a trusted environment and there is a strong trust relationship between the client and the root certificate authority. First, the rootCA' PKC must be installed manually.

The screen shot demonstration below will explain how to install these certificates:



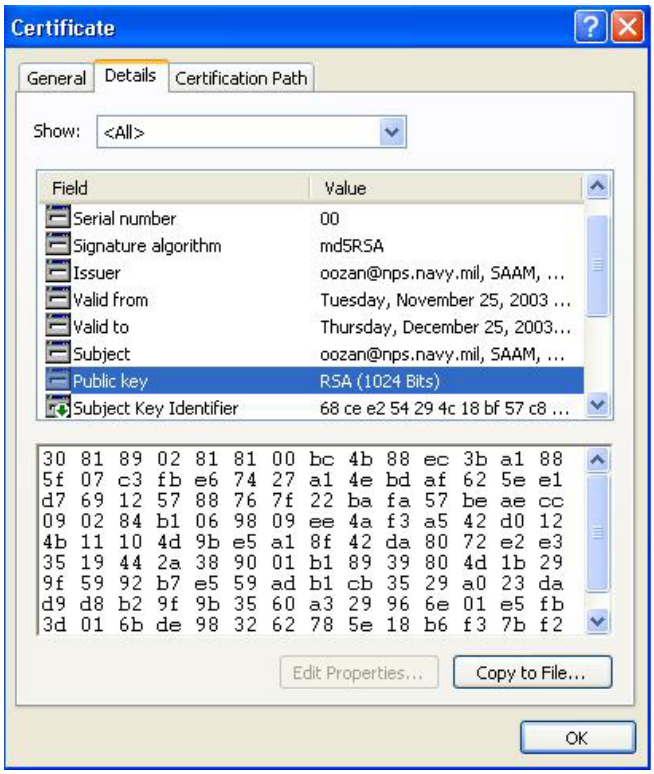
B.A1. When the root.der file is double-clicked, the certificate information window comes up. Then the "Install Certificate" button must be clicked.

B.A2. The "Place all certificate in the following store" radio button must be checked and the "Trusted Root Certification Authorities" must be highlighted. Then click OK.

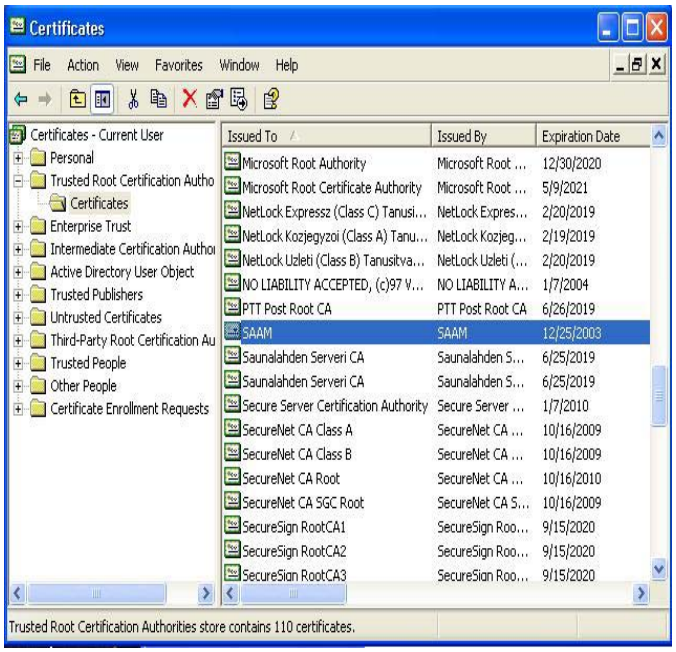
B.A3. Click Next to continue

B.A4. Click “Finish” to complete the root certificate import wizard

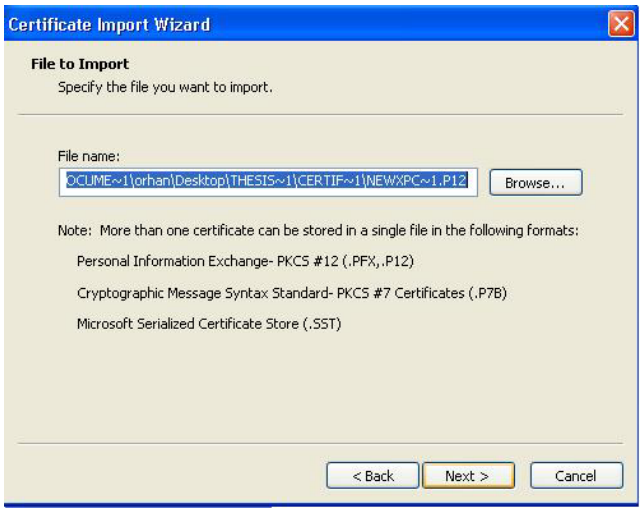
B.A5. If everything has been right. The system responds with the “successful import window



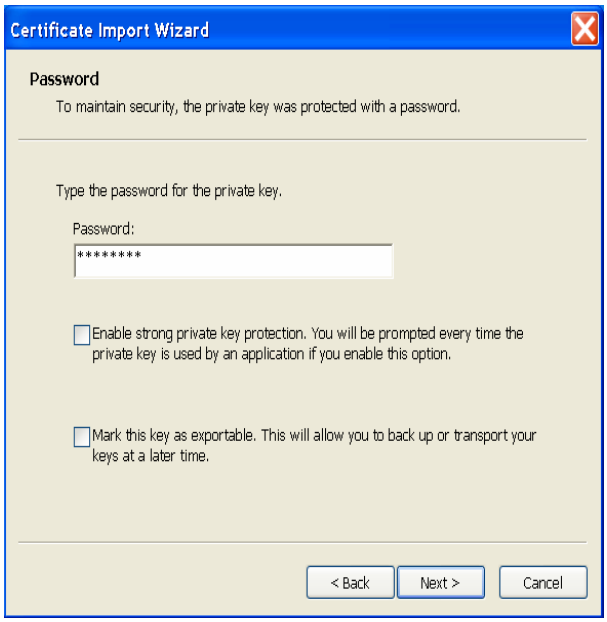
B.A6. 1024 bits RSA public key and other properties of the root certificate can be monitored through the “details” tab



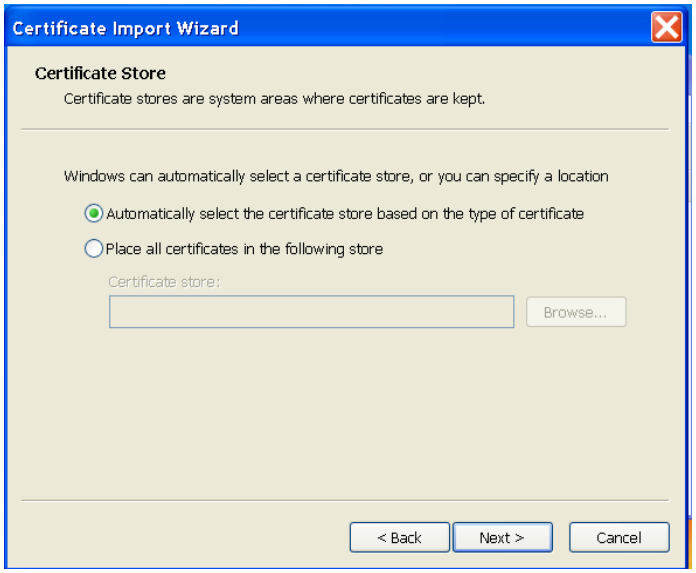
B.A7. The rootCA PKC can be verified via the MMC console whether it is under Trusted Certificate Authorities or not.



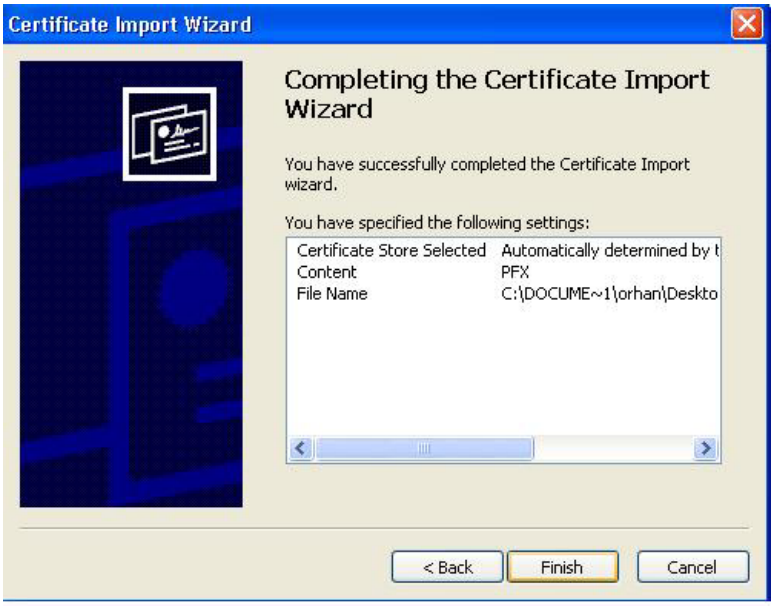
B.A8. When the newxpclient.p12 file is double-clicked, the certificate installation wizard appears on the screen. Click "Next" to continue.



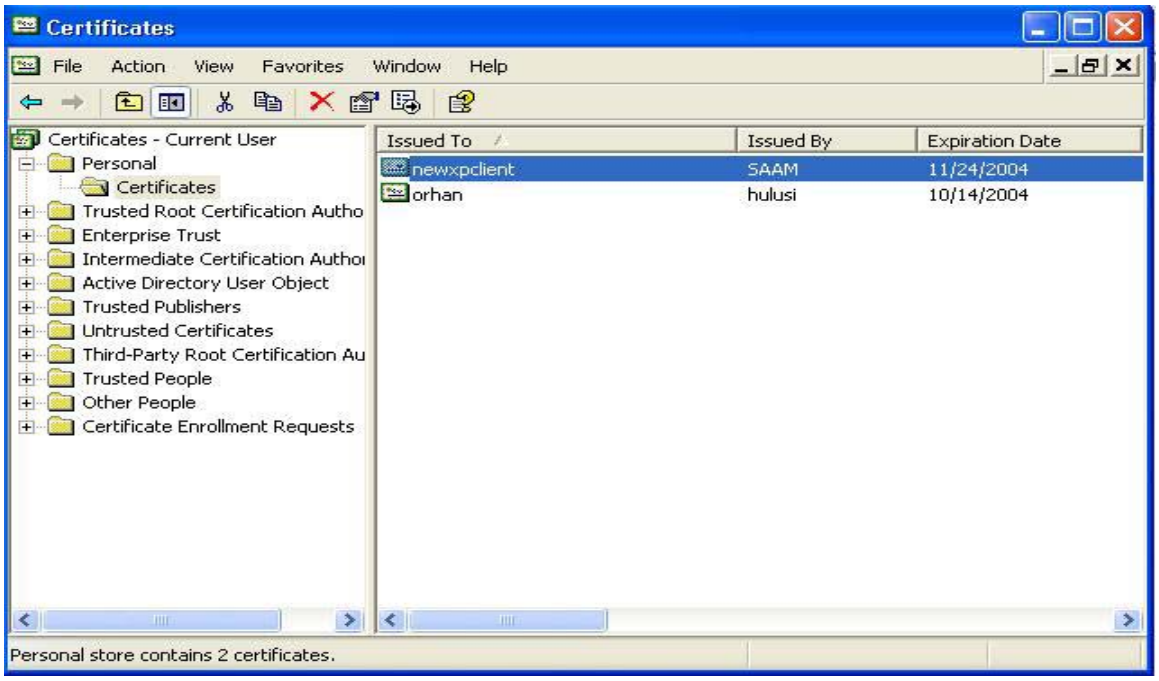
B.A9. The password to decrypt the private key for the client must be entered correctly. The password can be obtained manually from the certificate manager. Then click "Next" to continue.



B.A10 Leave the default values and click next

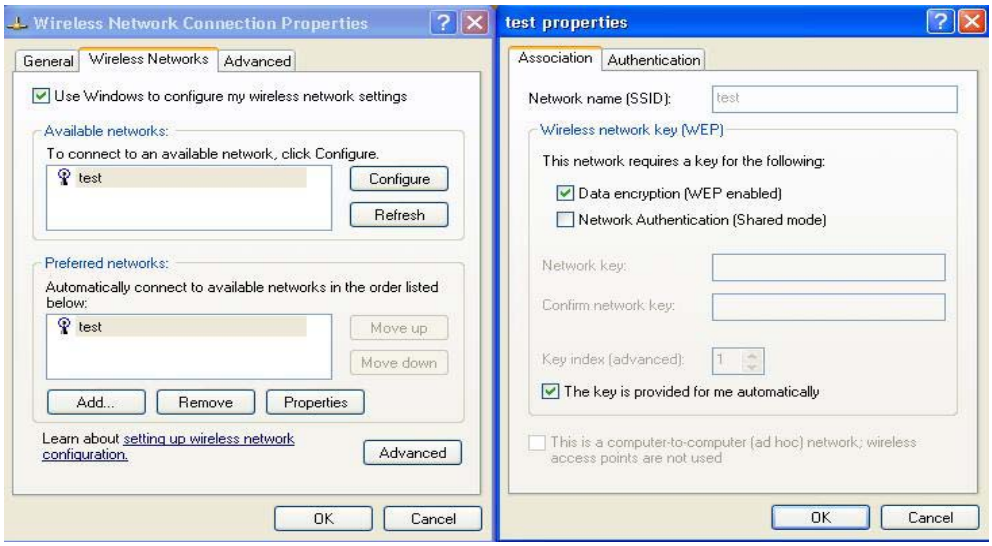


B.A11. Click “Finish” for successful client certificate import

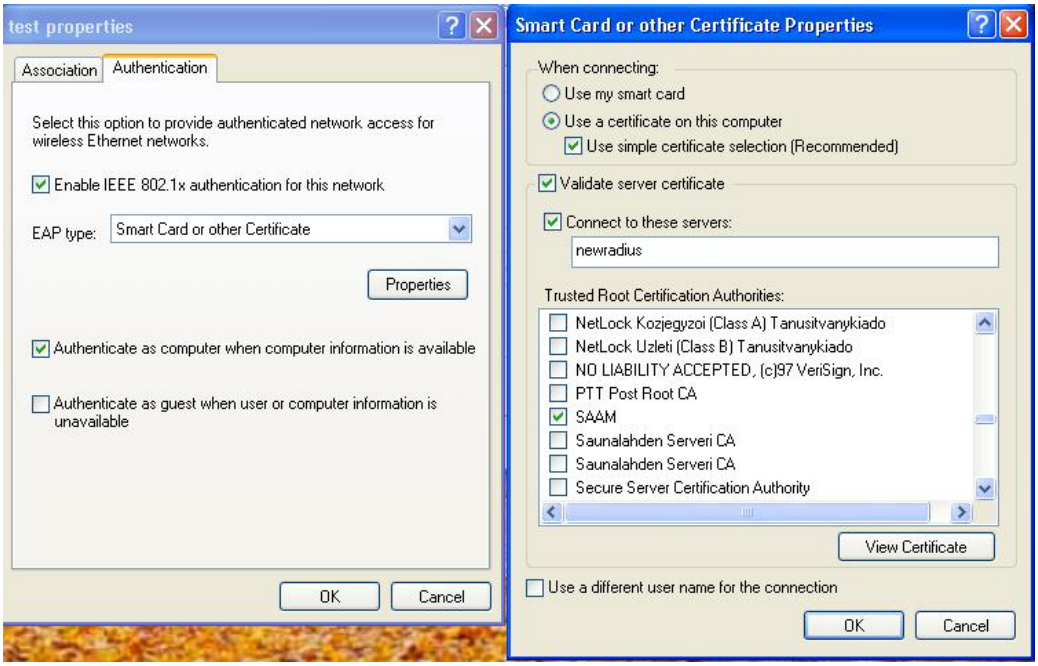


B.A12. The certification path should be verified under in the MMC window.

B. WINDOWS XP WIRELESS CLIENT 802.1X CONFIGURATION



B.B1. The WEP and dynamic key options must be checked in order to support the dynamic key generation from the authenticator.

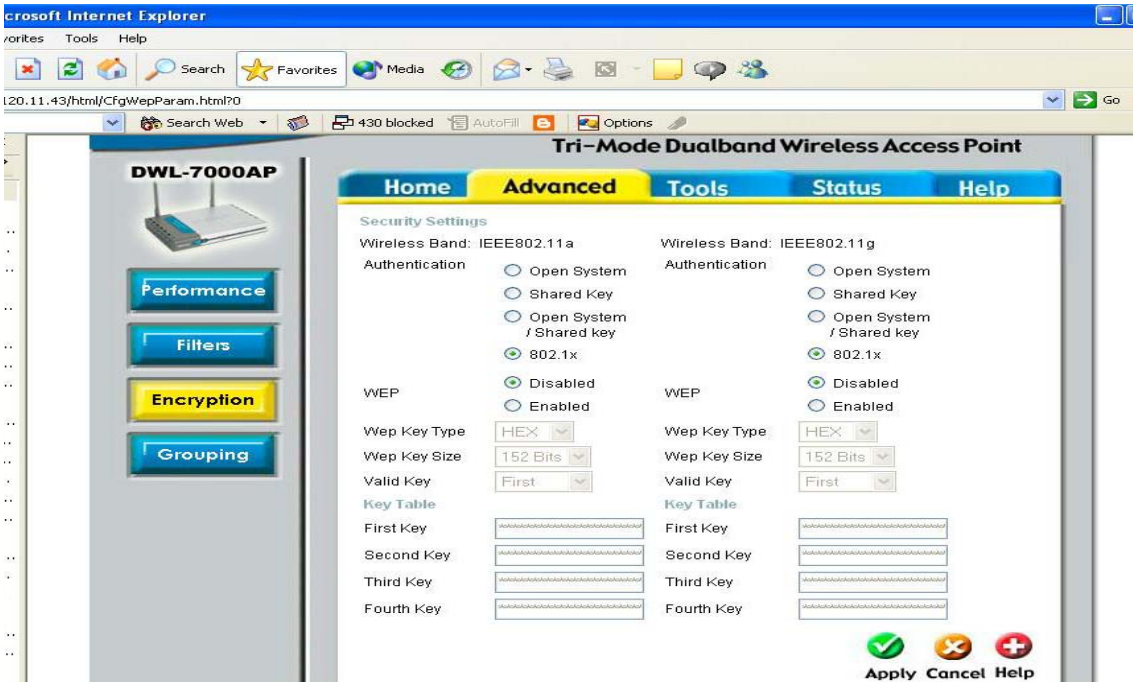


B.B2 The "Enable IEEE 802.1X authentication for this network" radio button must be checked with the "Smart Card or other Certificates" option. When the "Properties" button is clicked, the "Use a certificate on this computer" radio button must be checked. The "Validate server certificate" radio button must be checked; otherwise, only the client certificate is validated at the server. The server certificate must also be validated to enforce mutual authentication.

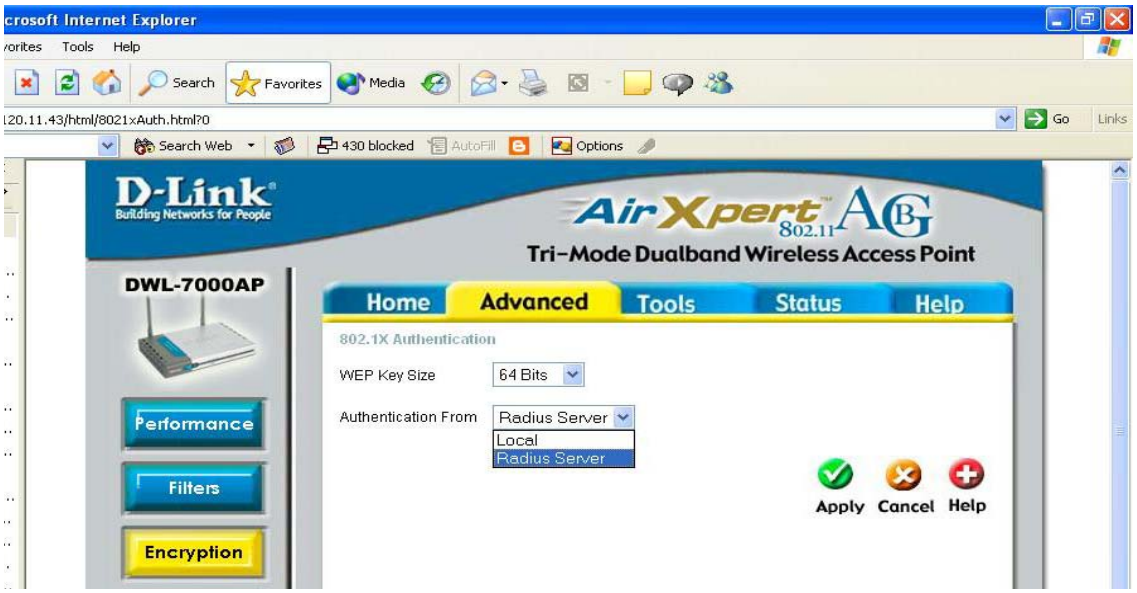
APPENDIX C

A. D-LINK DWL-7000AP CONFIGURATION

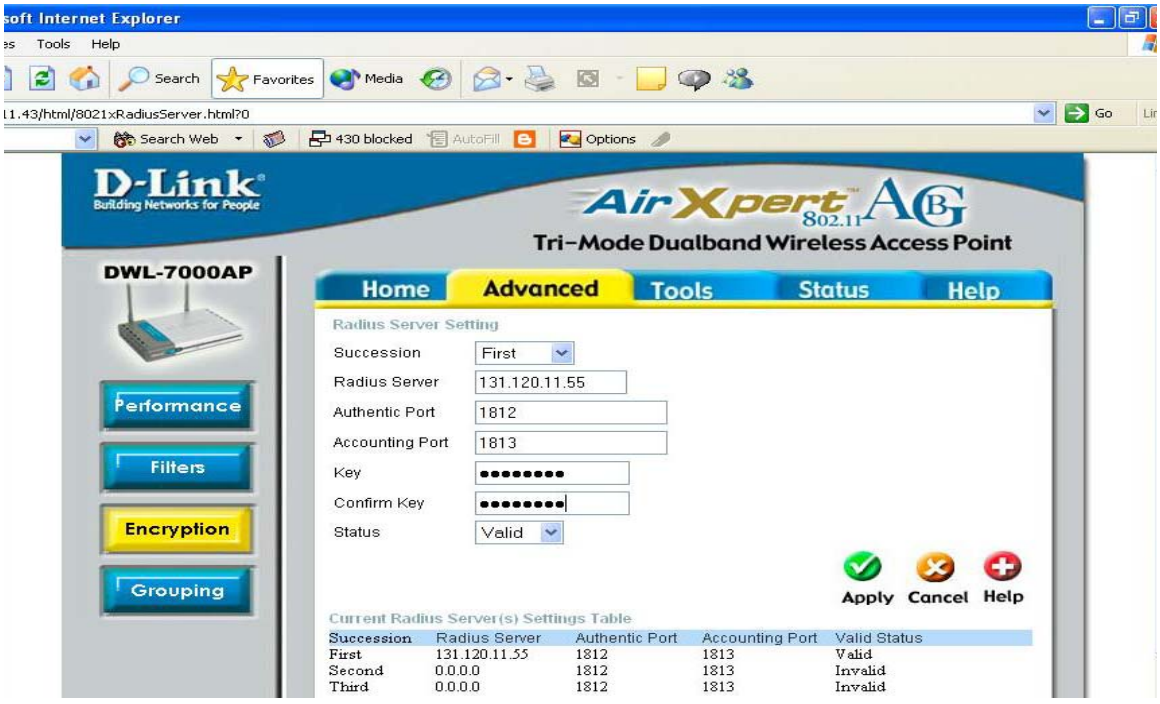
C.A1 D-link DWL-7000AP comes with an access point manager. This manager is useful to identify and discover the access points in the network. After discovering the access point, a new IP address can be assigned.



C.A2 After assigning a legitimate IP address to the access point, a web browser is used to configure the access point. The IP address of the AP is written to the address window and the web-based configuration tool is ready to be used. 802.1X option can be selected under *advanced* tab and *encryption* page.



C.A3 The next page after selecting the 802.1X option is the authentication server selection page. Radius server is selected from the drop down menu.



C.A4 the next page after selecting the radius server helps the user apply the specifications of the authentication server e.g. IP address, authentication port and the shared Key. Once this fields are completed, the AP is restarted and able to serve as an authenticator.

B. HOSTAP CONFIGURATION FILE

```
##### hostapd configuration file
##############################################
# Empty lines and lines starting with # are ignored

# AP netdevice name (without 'ap' prefix, i.e., wlan0 uses wlan0ap for
# management frames)
interface=wlan0

# Debugging: 0 = no, 1 = minimal, 2 = verbose, 3 = msg dumps
debug=3

# Dump file for state information (on SIGUSR1)
dump_file=/tmp/hostapd.dump

# Daemonize hostapd process (i.e., fork to background)
daemonize=1

##### IEEE 802.11 related configuration
#########################################

# SSID to be used in IEEE 802.11 management frames
ssid=test

# Station MAC address -based authentication
# 0 = accept unless in deny list
# 1 = deny unless in accept list
# 2 = use external RADIUS server (accept/deny lists are searched first)
macaddr_acl=0

# Accept/deny lists are read from separate files (containing list of
# MAC addresses, one per line). Use absolute path name to make sure
# that the files can be read on SIGHUP configuration reloads.
#accept_mac_file=/etc/hostapd.accept
#deny_mac_file=/etc/hostapd.deny

# Associate as a station to another AP while still acting as an AP on
# the same channel.
#assoc_ap_addr=00:12:34:56:78:9a


##### IEEE 802.1X (and IEEE 802.1aa/D4) related configuration #################

# Require IEEE 802.1X authorization
ieee8021x=1
```

```
# Use internal minimal EAP Authentication Server for testing IEEE 802.1X.
# This should only be used for testing since it authorizes all users
# that suppot IEEE 802.1X without any keys or certificates.
minimal_eap=0

# Optional displayable message sent with EAP Request-Identity
eap_message=hello

# WEP rekeying (disabled if key lengths are not set or are set to 0)
# Key lengths for default/broadcast and individual/unicast keys:
# 5 = 40-bit WEP (also known as 64-bit WEP with 40 secret bits)
# 13 = 104-bit WEP (also known as 128-bit WEP with 104 secret bits)
wep_key_len_broadcast=5
wep_key_len_unicast=5
#Rekeying period in seconds. 0 = do not rekey (i.e., set keys only once)
wep_rekey_period=300

# EAPOL-Key index workaround (set bit7) for WinXP Supplicant (needed only if
# only broadcast keys are used)
eapol_key_index_workaround=1

##### IEEE 802.11f - Inter-Access Point Protocol (IAPP) ######################

# Interface to be used for IAPP broadcast packets
#iapp_interface=eth0

##### RADIUS configuration
#########################################################
# for IEEE 802.1X with external Authentication Server, IEEE 802.11
# authentication with external ACL for MAC addresses, and accounting

# The own IP address of the access point (used as NAS-IP-Address)
own_ip_addr=131.120.8.145

# RADIUS authentication server
auth_server_addr=131.120.11.55
auth_server_port=1812
auth_server_shared_secret=besiktas

# RADIUS accounting server
#acct_server_addr=127.0.0.1
#acct_server_port=1813
#acct_server_shared_secret=secret
```

APPENDIX D

A. FREERADIUS EAP-TLS MODULE MAKE FILE

```
# Generated automatically from Makefile.in by configure.
TARGET      = rlm_eap_tls
SRCS        = rlm_eap_tls.c eap_tls.c cb.c tls.c mppe_keys.c
RLM_CFLAGS  = $(INCLTDL) -I../..    -DOPENSSL_NO_KRB5
HEADERS     = rlm_eap_tls.h eap_tls.h ../../eap.h ../../rlm_eap.h
RLM_INSTALL =
RLM_LIBS    +=  -lcrypto -lssl

$(STATIC_OBJS): $(HEADERS)

$(DYNAMIC_OBJS): $(HEADERS)

RLM_DIR=../../
include ${RLM_DIR}../rules.mak
```

B. RADIUSD CONFIGURATION FILE

(To save space, the comments are omitted)

```
prefix = /usr/local
exec_prefix = ${prefix}
sysconfdir = /etc
localstatedir = ${prefix}/var
sbindir = ${exec_prefix}/sbin
logdir = ${localstatedir}/log/radius
raddbdir = ${sysconfdir}/raddb
radacctdir = ${logdir}/radacct

confdir = ${raddbdir}
run_dir = ${localstatedir}/run/radiusd

log_file = ${logdir}/radius.log

libdir = ${exec_prefix}/lib

pidfile = ${run_dir}/radiusd.pid


max_request_time = 30

delete_blocked_requests = no
```

```
cleanup_delay = 5

max_requests = 1024

bind_address = *

port = 0

hostname_lookups = no

allow_core_dumps = no

regular_expressions    = yes
extended_expressions = yes

log_stripped_names = no

log_auth = no

log_auth_badpass = no
log_auth_goodpass = no

usercollide = no

lower_user = no
lower_pass = no

nospace_user = no
nospace_pass = no

checkrad = ${sbindir}/checkrad

# SECURITY CONFIGURATION
security {
        max_attributes = 200
        reject_delay = 1
        status_server = no
}

# PROXY CONFIGURATION

proxy_requests  = yes

$INCLUDE  ${confdir}/proxy.conf
```

```
# CLIENTS CONFIGURATION

$INCLUDE  ${confdir}/clients.conf


# SNMP CONFIGURATION
snmp   = no
$INCLUDE  ${confdir}/snmp.conf


# THREAD POOL CONFIGURATION
thread pool {
        start_servers = 5

        max_servers = 32

        min_spare_servers = 3
        max_spare_servers = 10

        max_requests_per_server = 0
}

# MODULE CONFIGURATION
modules {
        pap {
                encryption_scheme = crypt
        }

        # CHAP module
        chap {
                authtype = CHAP
        }

        # Pluggable Authentication Modules
        pam {
                pam_auth = radiusd
        }

        unix {
                cache = yes


                cache_reload = 600
```

```
                passwd = /etc/passwd
                shadow = /etc/shadow
                group = /etc/group

        radwtmp = ${logdir}/radwtmp
}

eap {
        default_eap_type = tls
        timer_expire     = 60

        ignore_unknown_eap_types = no


        tls {
                private_key_password = whatever
                private_key_file = /etc/1x/newradius.pem

                certificate_file = /etc/1x/newradius.pem

                CA_file = /etc/1x/root.pem

                dh_file = /etc/1x/DH
                random_file = /etc/1x/random

                fragment_size = 1024

                include_length = yes
        #       check_crl = yes
        }

        mschapv2 {
        }
}

mschap {

        authtype = MS-CHAP


}

ldap {
        server = "ldap.your.domain"
        basedn = "o=My Org,c=UA"
        filter = "(uid=%{Stripped-User-Name:-%{User-Name}})"
```

```
        start_tls = no

        access_attr = "dialupAccess"

        dictionary_mapping = ${raddbdir}/ldap.attrmap

        ldap_connections_number = 5

        timeout = 4
        timelimit = 3
        net_timeout = 1
}

realm realmslash {
        format = prefix
        delimiter = "/"
}

#  'username@realm'
#
realm suffix {
        format = suffix
        delimiter = "@"
}

#  'username%realm'
#
realm realmpercent {
        format = suffix
        delimiter = "%"
}

preprocess {
        huntgroups = ${confdir}/huntgroups
        hints = ${confdir}/hints

        with_ascend_hack = no
        ascend_channels_per_line = 23

        with_ntdomain_hack = no

        with_specialix_jetstream_hack = no

        with_cisco_vsa_hack = no
}
```

```
        files {
                usersfile = ${confdir}/users
                acctusersfile = ${confdir}/acct_users

                compat = no
        }

        detail {
                detailfile = ${radacctdir}/%{Client-IP-Address}/detail-%Y%m%d

                detailperm = 0600
        }

        acct_unique {
                key = "User-Name, Acct-Session-Id, NAS-IP-Address, Client-IP-Address,
NAS-Port-Id"
        }


        $INCLUDE  ${confdir}/sql.conf

        radutmp {
                filename = ${logdir}/radutmp

                username = %{User-Name}

                case_sensitive = yes

                check_with_nas = yes
                perm = 0600

                callerid = "yes"
        }

        radutmp sradutmp {
                filename = ${logdir}/sradutmp
                perm = 0644
                callerid = "no"
        }

        attr_filter {
                attrsfile = ${confdir}/attrs
        }

        counter daily {
                filename = ${raddbdir}/db.daily
```

```
        key = User-Name
        count-attribute = Acct-Session-Time
        reset = daily
        counter-name = Daily-Session-Time
        check-name = Max-Daily-Session
        allowed-servicetype = Framed-User
        cache-size = 5000
}

always fail {
        rcode = fail
}
always reject {
        rcode = reject
}
always ok {
        rcode = ok
        simulcount = 0
        mpp = no
}

expr {
}

digest {
}

exec {
        wait = yes
        input_pairs = request
}

exec echo {
        wait = yes

        program = "/bin/echo %{User-Name}"

        input_pairs = request

        output_pairs = reply

}
ippool main_pool {


        netmask = 255.255.255.0
```

```
                cache-size = 800

                session-db = ${raddbdir}/db.ippool

                ip-index = ${raddbdir}/db.ipindex

                override = no
        }


}

instantiate {
        expr

}

authorize {
        preprocess

        eap

        suffix
        files




}


authenticate {

        unix



}


preacct {
        preprocess

        suffix
```

```
}

accounting {
        acct_unique

        detail

        unix            # wtmp file

        radutmp

}


session {
        radutmp
}


post-auth {

}

pre-proxy {
}

post-proxy {


        eap
}
```

C. CLIENTS CONFIGURATION FILE

```
#
# clients.conf - client configuration directives
#
# This file is included by default.  To disable it, you will need
# to modify the CLIENTS CONFIGURATION section of "radiusd.conf".
#
#########################################################################

#########################################################################
#
#  Definition of a RADIUS client (usually a NAS).
#
#  The information given here over rides anything given in the 'clients'
#  file, or in the 'naslist' file.  The configuration here contains
#  all of the information from those two files, and also allows for more
#  configuration items.
#
#  The "shortname" can be used for logging, and the "nastype",
#  "login" and "password" fields are mainly used for checkrad and are
#  optional.
#

#
#  Defines a RADIUS client.  The format is 'client [hostname|ip-address]'
#
#  '127.0.0.1' is another name for 'localhost'.  It is enabled by default,
#  to allow testing of the server after an initial installation.  If you
#  are not going to be permitting RADIUS queries from localhost, we suggest
#  that you delete, or comment out, this entry.
#
################################Orhan ekledi#########################
#client 131.120.10.153 {
#secret = whatever
#shortname = CLIENT1
#}

client 131.120.8.145 {
secret = besiktas
shortname = AP1
}
#client 131.120.10.133 {
#secret = wahtever
#shortname = AP2
#}
#########################################################################
```

```
client 127.0.0.1 {
        #
        #  The shared secret use to "encrypt" and "sign" packets between
        #  the NAS and FreeRADIUS.  You MUST change this secret from the
        #  default, otherwise it's not a secret any more!
        #
        #  The secret can be any string, up to 32 characters in length.
        #
        secret          = test

        #
        #  The short name is used as an alias for the fully qualified
        #  domain name, or the IP address.
        #
        shortname       = localhost

        #
        # the following three fields are optional, but may be used by
        # checkrad.pl for simultaneous use checks
        #

        #
        # The nastype tells 'checkrad.pl' which NAS-specific method to
        #  use to query the NAS for simultaneous use.
        #
        #  Permitted NAS types are:
        #
        #       cisco
        #       computone
        #       livingston
        #       max40xx
        #       multitech
        #       netserver
        #       pathras
        #       patton
        #       portslave
        #       tc
        #       usrhiper
        #       other           # for all other types

        #
        nastype     = other     # localhost isn't usually a NAS...

        #
        #  The following two configurations are for future use.
```

```
        #  The 'naspasswd' file is currently used to store the NAS
        #  login name and password, which is used by checkrad.pl
        #  when querying the NAS for simultaneous use.
        #
#       login      = !root
#       password   = someadminpas
}

#client some.host.org {
#       secret          = testing123
#       shortname       = localhost
#}

#
#  You can now specify one secret for a network of clients.
#  When a client request comes in, the BEST match is chosen.
#  i.e. The entry from the smallest possible network.
#
#client 192.168.0.0/24 {
#       secret          = testing123-1
#       shortname       = private-network-1
#}
#
#client 192.168.0.0/16 {
#       secret          = testing123-2
#       shortname       = private-network-2
#}


#client 10.10.10.10 {
#       # secret and password are mapped through the "secrets" file.
#       secret      = testing123
#       shortname   = liv1
#     # the following three fields are optional, but may be used by
#     # checkrad.pl for simultaneous usage checks
#       nastype     = livingston
#       login       = !root
#       password    = someadminpas
#}
```

D. USERS CONFIGURATION FILE

```
#
#       Please read the documentation file ../doc/processing_users_file,
#       or 'man 5 users' (after installing the server) for more information.
#
```

```
#       This file contains authentication security and configuration
#       information for each user.  Accounting requests are NOT processed
#       through this file.  Instead, see 'acct_users', in this directory.
#
#       The first field is the user's name and can be up to
#       253 characters in length.  This is followed (on the same line) with
#       the list of authentication requirements for that user.  This can
#       include password, comm server name, comm server port number, protocol
#       type (perhaps set by the "hints" file), and huntgroup name (set by
#       the "huntgroups" file).
#
#       If you are not sure why a particular reply is being sent by the
#       server, then run the server in debugging mode (radiusd -X), and
#       you will see which entries in this file are matched.
#
#       When an authentication request is received from the comm server,
#       these values are tested. Only the first match is used unless the
#       "Fall-Through" variable is set to "Yes".
#
#       A special user named "DEFAULT" matches on all usernames.
#       You can have several DEFAULT entries. All entries are processed
#       in the order they appear in this file. The first entry that
#       matches the login-request will stop processing unless you use
#       the Fall-Through variable.
#
#       If you use the database support to turn this file into a .db or .dbm
#       file, the DEFAULT entries _have_ to be at the end of this file and
#       you can't have multiple entries for one username.
#
#       You don't need to specify a password if you set Auth-Type += System
#       on the list of authentication requirements. The RADIUS server
#       will then check the system password file.
#
#       Indented (with the tab character) lines following the first
#       line indicate the configuration values to be passed back to
#       the comm server to allow the initiation of a user session.
#       This can include things like the PPP configuration values
#       or the host to log the user onto.
#
#       You can include another `users' file with `$INCLUDE users.other'
#


#
#       For a list of RADIUS attributes, and links to their definitions,
#       see:
#
```

```
#       http://www.freeradius.org/rfc/attributes.html
#

#
# Deny access for a specific user.  Note that this entry MUST
# be before any other 'Auth-Type' attribute which results in the user
# being authenticated.
#
# Note that there is NO 'Fall-Through' attribute, so the user will not
# be given any additional resources.
#
#lameuser       Auth-Type := Reject
#               Reply-Message = "Your account has been disabled."

#
# Deny access for a group of users.
#
# Note that there is NO 'Fall-Through' attribute, so the user will not
# be given any additional resources.
#
#DEFAULT    Group == "disabled", Auth-Type := Reject
#               Reply-Message = "Your account has been disabled."
#

#
# This is a complete entry for "steve". Note that there is no Fall-Through
# entry so that no DEFAULT entry will be used, and the user will NOT
# get any attributes in addition to the ones listed here.
#
#steve  Auth-Type := Local, User-Password == "testing"
#       Service-Type = Framed-User,
#       Framed-Protocol = PPP,
#       Framed-IP-Address = 172.16.3.33,
#       Framed-IP-Netmask = 255.255.255.0,
#       Framed-Routing = Broadcast-Listen,
#       Framed-Filter-Id = "std.ppp",
#       Framed-MTU = 1500,
#       Framed-Compression = Van-Jacobsen-TCP-IP

#
# This is an entry for a user with a space in their name.
# Note the double quotes surrounding the name.
#
#"John Doe"    Auth-Type := Local, User-Password == "hello"
#               Reply-Message = "Hello, %u"
```

```
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
#!!!!!!!!!!!added by the designers!!!!!!!!!!!!!!!!!!!!!
#
newxpclient Auth-Type := EAP

test	Auth-Type := Local, User-Password == "test"
	Reply-Message = "hello,%u"
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
#
# Dial user back and telnet to the default host for that port
#
#Deg	Auth-Type := Local, User-Password == "ge55ged"
#	Service-Type = Callback-Login-User,
#	Login-IP-Host = 0.0.0.0,
#	Callback-Number = "9,5551212",
#	Login-Service = Telnet,
#	Login-TCP-Port = Telnet

#
# Another complete entry. After the user "dialbk" has logged in, the
# connection will be broken and the user will be dialed back after which
# he will get a connection to the host "timeshare1".
#
#dialbkAuth-Type := Local, User-Password == "callme"
#	Service-Type = Callback-Login-User,
#	Login-IP-Host = timeshare1,
#	Login-Service = PortMaster,
#	Callback-Number = "9,1-800-555-1212"

#
# user "swilson" will only get a static IP number if he logs in with
# a framed protocol on a terminal server in Alphen (see the huntgroups file).
#
# Note that by setting "Fall-Through", other attributes will be added from
# the following DEFAULT entries
#
#swilson		Service-Type == Framed-User, Huntgroup-Name == "alphen"
#		Framed-IP-Address = 192.168.1.65,
#		Fall-Through = Yes

#
# If the user logs in as 'username.shell', then authenticate them
# against the system database, give them shell access, and stop processing
# the rest of the file.
#
```

```
#DEFAULT    Suffix == ".shell", Auth-Type := System
#                   Service-Type = Login-User,
#                   Login-Service = Telnet,
#                   Login-IP-Host = your.shell.machine


#
# The rest of this file contains the several DEFAULT entries.
# DEFAULT entries match with all login names.
# Note that DEFAULT entries can also Fall-Through (see first entry).
# A name-value pair from a DEFAULT entry will _NEVER_ override
# an already existing name-value pair.
#

#
# First setup all accounts to be checked against the UNIX /etc/passwd.
# (Unless a password was already given earlier in this file).
#
DEFAULT     Auth-Type := System
        Fall-Through = 1



#
# Set up different IP address pools for the terminal servers.
# Note that the "+" behind the IP address means that this is the "base"
# IP address. The Port-Id (S0, S1 etc) will be added to it.
#
#DEFAULT    Service-Type == Framed-User, Huntgroup-Name == "alphen"
#                   Framed-IP-Address = 192.168.1.32+,
#                   Fall-Through = Yes

#DEFAULT    Service-Type == Framed-User, Huntgroup-Name == "delft"
#                   Framed-IP-Address = 192.168.2.32+,
#                   Fall-Through = Yes


#
# Defaults for all framed connections.
#
DEFAULT     Service-Type == Framed-User
        Framed-IP-Address = 255.255.255.254,
        Framed-MTU = 576,
        Service-Type = Framed-User,
        Fall-Through = Yes
```

```
#
# Default for PPP: dynamic IP address, PPP mode, VJ-compression.
# NOTE: we do not use Hint = "PPP", since PPP might also be auto-detected
#       by the terminal server in which case there may not be a "P" suffix.
#       The terminal server sends "Framed-Protocol = PPP" for auto PPP.
#
DEFAULT     Framed-Protocol == PPP
        Framed-Protocol = PPP,
        Framed-Compression = Van-Jacobson-TCP-IP

#
# Default for CSLIP: dynamic IP address, SLIP mode, VJ-compression.
#
DEFAULT     Hint == "CSLIP"
        Framed-Protocol = SLIP,
        Framed-Compression = Van-Jacobson-TCP-IP

#
# Default for SLIP: dynamic IP address, SLIP mode.
#
DEFAULT     Hint == "SLIP"
        Framed-Protocol = SLIP

#
# Last default: rlogin to our main server.
#
#DEFAULT
#       Service-Type = Login-User,
#       Login-Service = Rlogin,
#       Login-IP-Host = shellbox.ispdomain.com

# #
# # Last default: shell on the local terminal server.
# #
# DEFAULT
#       Service-Type = Shell-User

# On no match, the user is denied access.
```

E. RADIUSD RUNNING SCRIPT

```
#!/bin/sh -x

LD_LIBRARY_PATH=/usr/local/openssl/lib
LD_PRELOAD=/usr/local/openssl/lib/libcrypto.so

export LD_LIBRARY_PATH LD_PRELOAD

/usr/local/sbin/radiusd -X -A $@
```

APPENDIX E

A. AUTHENTICATION SERVER SUCCESSFUL AUTHENTICATION LOGS

```
Starting - reading configuration files ...
reread_config:  reading radiusd.conf
Config:   including file: /etc/raddb/proxy.conf
Config:   including file: /etc/raddb/clients.conf
Config:   including file: /etc/raddb/snmp.conf
Config:   including file: /etc/raddb/sql.conf
 main: prefix = "/usr/local"
 main: localstatedir = "/usr/local/var"
 main: logdir = "/usr/local/var/log/radius"
 main: libdir = "/usr/local/lib"
 main: radacctdir = "/usr/local/var/log/radius/radacct"
 main: hostname_lookups = no
 main: max_request_time = 30
 main: cleanup_delay = 5
 main: max_requests = 1024
 main: delete_blocked_requests = 0
 main: port = 0
 main: allow_core_dumps = no
 main: log_stripped_names = no
 main: log_file = "/usr/local/var/log/radius/radius.log"
 main: log_auth = no
 main: log_auth_badpass = no
 main: log_auth_goodpass = no
 main: pidfile = "/usr/local/var/run/radiusd/radiusd.pid"
 main: user = "(null)"
 main: group = "(null)"
 main: usercollide = no
 main: lower_user = "no"
 main: lower_pass = "no"
 main: nospace_user = "no"
 main: nospace_pass = "no"
 main: checkrad = "/usr/local/sbin/checkrad"
 main: proxy_requests = yes
 proxy: retry_delay = 5
 proxy: retry_count = 3
 proxy: synchronous = no
 proxy: default_fallback = yes
 proxy: dead_time = 120
 proxy: post_proxy_authorize = yes
 proxy: wake_all_if_all_dead = no
 security: max_attributes = 200
 security: reject_delay = 1
 security: status_server = no
 main: debug_level = 0
read_config_files:  reading dictionary
read_config_files:  reading naslist
Using deprecated naslist file.  Support for this will go away soon.
read_config_files:  reading clients
Using deprecated clients file.  Support for this will go away soon.
read_config_files:  reading realms
```

```
Using deprecated realms file.  Support for this will go away soon.
radiusd:  entering modules setup
Module: Library search path is /usr/local/lib
Module: Loaded expr
Module: Instantiated expr (expr)
Module: Loaded System
 unix: cache = yes
 unix: passwd = "/etc/passwd"
 unix: shadow = "/etc/shadow"
 unix: group = "/etc/group"
 unix: radwtmp = "/usr/local/var/log/radius/radwtmp"
 unix: usegroup = no
 unix: cache_reload = 600
HASH:  Reinitializing hash structures and lists for caching...
  HASH:  user root found in hashtable bucket 11726
  HASH:  user bin found in hashtable bucket 86651
  HASH:  user daemon found in hashtable bucket 11668
  HASH:  user adm found in hashtable bucket 26466
  HASH:  user lp found in hashtable bucket 54068
  HASH:  user sync found in hashtable bucket 42895
  HASH:  user shutdown found in hashtable bucket 71746
  HASH:  user halt found in hashtable bucket 7481
  HASH:  user mail found in hashtable bucket 79471
  HASH:  user news found in hashtable bucket 5375
  HASH:  user uucp found in hashtable bucket 38541
  HASH:  user operator found in hashtable bucket 21748
  HASH:  user games found in hashtable bucket 47657
  HASH:  user gopher found in hashtable bucket 47357
  HASH:  user ftp found in hashtable bucket 56226
  HASH:  user nobody found in hashtable bucket 99723
  HASH:  user rpm found in hashtable bucket 72383
  HASH:  user vcsa found in hashtable bucket 25959
  HASH:  user nscd found in hashtable bucket 36306
  HASH:  user sshd found in hashtable bucket 71560
  HASH:  user rpc found in hashtable bucket 72373
  HASH:  user rpcuser found in hashtable bucket 552
  HASH:  user nfsnobody found in hashtable bucket 51830
  HASH:  user mailnull found in hashtable bucket 78086
  HASH:  user smmsp found in hashtable bucket 13600
  HASH:  user pcap found in hashtable bucket 55326
  HASH:  user xfs found in hashtable bucket 17213
  HASH:  user ntp found in hashtable bucket 21418
  HASH:  user gdm found in hashtable bucket 50360
  HASH:  user oozan found in hashtable bucket 94479
  HASH:  user amanda found in hashtable bucket 72438
HASH:  Stored 31 entries from /etc/passwd
HASH:  Stored 39 entries from /etc/group
Module: Instantiated unix (unix)
Module: Loaded eap
 eap: default_eap_type = "tls"
 eap: timer_expire = 60
 eap: ignore_unknown_eap_types = no
 tls: rsa_key_exchange = no
 tls: dh_key_exchange = yes
 tls: rsa_key_length = 512
 tls: dh_key_length = 512
 tls: verify_depth = 0
```

```
 tls: CA_path = "(null)"
 tls: pem_file_type = yes
 tls: private_key_file = "/etc/1x/newradius.pem"
 tls: certificate_file = "/etc/1x/newradius.pem"
 tls: CA_file = "/etc/1x/root.pem"
 tls: private_key_password = "whatever"
 tls: dh_file = "/etc/1x/DH"
 tls: random_file = "/etc/1x/random"
 tls: fragment_size = 1024
 tls: include_length = yes
 tls: check_crl = no
rlm_eap: Loaded and initialized type tls
rlm_eap: Loaded and initialized type mschapv2
Module: Instantiated eap (eap)
Module: Loaded preprocess
 preprocess: huntgroups = "/etc/raddb/huntgroups"
 preprocess: hints = "/etc/raddb/hints"
 preprocess: with_ascend_hack = no
 preprocess: ascend_channels_per_line = 23
 preprocess: with_ntdomain_hack = no
 preprocess: with_specialix_jetstream_hack = no
 preprocess: with_cisco_vsa_hack = no
Module: Instantiated preprocess (preprocess)
Module: Loaded realm
 realm: format = "suffix"
 realm: delimiter = "@"
Module: Instantiated realm (suffix)
Module: Loaded files
 files: usersfile = "/etc/raddb/users"
 files: acctusersfile = "/etc/raddb/acct_users"
 files: preproxy_usersfile = "/etc/raddb/preproxy_users"
 files: compat = "no"
Module: Instantiated files (files)
Module: Loaded Acct-Unique-Session-Id
 acct_unique: key = "User-Name, Acct-Session-Id, NAS-IP-Address,
Client-IP-Address, NAS-Port-Id"
Module: Instantiated acct_unique (acct_unique)
Module: Loaded detail
 detail: detailfile = "/usr/local/var/log/radius/radacct/%{Client-IP-
Address}/detail-%Y%m%d"
 detail: detailperm = 384
 detail: dirperm = 493
 detail: locking = no
Module: Instantiated detail (detail)
Module: Loaded radutmp
 radutmp: filename = "/usr/local/var/log/radius/radutmp"
 radutmp: username = "%{User-Name}"
 radutmp: case_sensitive = yes
 radutmp: check_with_nas = yes
 radutmp: perm = 384
 radutmp: callerid = yes
Module: Instantiated radutmp (radutmp)
Listening on IP address *, ports 1812/udp and 1813/udp, with proxy on
1814/udp.
Ready to process requests.
rad_recv: Access-Request packet from host 131.120.8.145:32804, id=0,
length=160
```

```
      User-Name = "newxpclient"
      NAS-IP-Address = 131.120.8.145
      NAS-Port = 1
      Called-Station-Id = "00-05-5D-D9-8D-AE:test"
      Calling-Station-Id = "00-05-5D-D9-57-59"
      Framed-MTU = 2304
      NAS-Port-Type = Wireless-802.11
      Connect-Info = "CONNECT 11Mbps 802.11b"
      EAP-Message = "\002\001\000\020\001newxpclient"
      Message-Authenticator = 0x7b47883e05d44aa13d69442f35d1178f
modcall: entering group authorize for request 0
  modcall[authorize]: module "preprocess" returns ok for request 0
  rlm_eap: EAP packet type response id 1 length 16
  rlm_eap: No EAP Start, assuming it's an on-going EAP conversation
  modcall[authorize]: module "eap" returns updated for request 0
    rlm_realm: No '@' in User-Name = "newxpclient", looking up realm
NULL
    rlm_realm: No such realm "NULL"
  modcall[authorize]: module "suffix" returns noop for request 0
    users: Matched newxpclient at 101
  modcall[authorize]: module "files" returns ok for request 0
modcall: group authorize returns updated for request 0
  rad_check_password:  Found Auth-Type EAP
auth: type "EAP"
modcall: entering group authenticate for request 0
  rlm_eap: EAP Identity
  rlm_eap: processing type tls
 rlm_eap_tls: Requiring client certificate
  rlm_eap_tls: Initiate
  rlm_eap_tls: Start returned 1
  modcall[authenticate]: module "eap" returns handled for request 0
modcall: group authenticate returns handled for request 0
Sending Access-Challenge of id 0 to 131.120.8.145:32804
      EAP-Message = "\001\002\000\006\r "
      Message-Authenticator = 0x00000000000000000000000000000000
      State = 0xcabe64f58e2e52c0326344aeaf7ff16d
Finished request 0
-------
-------
-------
-------
Going to the next request
Waking up in 1 seconds...
rad_recv: Access-Request packet from host 131.120.8.145:32804, id=13,
length=168
      User-Name = "newxpclient"
      NAS-IP-Address = 131.120.8.145
      NAS-Port = 1
      Called-Station-Id = "00-05-5D-D9-8D-AE:test"
      Calling-Station-Id = "00-05-5D-D9-57-59"
      Framed-MTU = 2304
      NAS-Port-Type = Wireless-802.11
      Connect-Info = "CONNECT 11Mbps 802.11b"
      EAP-Message = "\002\020\000\006\r"
      State = 0x2c02871aafebad85e9d5602736783471
      Message-Authenticator = 0xb1f4cfdc73e426a656047670cc908ef6
modcall: entering group authorize for request 13
```

```
  modcall[authorize]: module "preprocess" returns ok for request 13
  rlm_eap: EAP packet type response id 16 length 6
  rlm_eap: No EAP Start, assuming it's an on-going EAP conversation
  modcall[authorize]: module "eap" returns updated for request 13
    rlm_realm: No '@' in User-Name = "newxpclient", looking up realm
NULL
    rlm_realm: No such realm "NULL"
  modcall[authorize]: module "suffix" returns noop for request 13
    users: Matched newxpclient at 101
  modcall[authorize]: module "files" returns ok for request 13
modcall: group authorize returns updated for request 13
  rad_check_password:  Found Auth-Type EAP
auth: type "EAP"
modcall: entering group authenticate for request 13
  rlm_eap: Request found, released from the list
  rlm_eap: EAP_TYPE - tls
  rlm_eap: processing type tls
  rlm_eap_tls: Authenticate
  rlm_eap_tls: processing TLS
rlm_eap_tls: Received EAP-TLS ACK message
  rlm_eap_tls: ack handshake is finished
  eaptls_verify returned 3
  eaptls_process returned 3
  rlm_eap: Freeing handler
  modcall[authenticate]: module "eap" returns ok for request 13
modcall: group authenticate returns ok for request 13
Sending Access-Accept of id 13 to 131.120.8.145:32804
      MS-MPPE-Recv-Key =
0xe9545b180975cdfb5d0f982189e03b43602f6c475e4ee66d7d24783c056f314c
      MS-MPPE-Send-Key =
0x1f44ce961ecf533c728e731dff144b918ed1a420fd83185e4ecc6b3c686a08b9
      EAP-Message = "\003\020\000\004"
      Message-Authenticator = 0x00000000000000000000000000000000
      User-Name = "newxpclient"
Finished request 13
Going to the next request
Waking up in 1 seconds...
--- Walking the entire request list ---
Waking up in 2 seconds...
--- Walking the entire request list ---
Cleaning up request 4 ID 4 with timestamp 4006d26e
Cleaning up request 5 ID 5 with timestamp 4006d26e
Cleaning up request 6 ID 6 with timestamp 4006d26e
Cleaning up request 7 ID 7 with timestamp 4006d26e
Cleaning up request 8 ID 8 with timestamp 4006d26e
Waking up in 3 seconds...
--- Walking the entire request list ---
Cleaning up request 9 ID 9 with timestamp 4006d271
Cleaning up request 10 ID 10 with timestamp 4006d271
Cleaning up request 11 ID 11 with timestamp 4006d271
Cleaning up request 12 ID 12 with timestamp 4006d271
Cleaning up request 13 ID 13 with timestamp 4006d271
Nothing to do.  Sleeping until we see a request.
```

B. AUTHENTICATOR SUCCESSFUL SUPPLICANT AUTHENTICATION LOG

```
Opening raw packet socket for ifindex 4
Using interface wlan0ap with hwaddr 00:05:5d:d9:8d:ae and ssid 'test'
Default WEP key - hexdump(len=5): 8a 0a ef db b7
Flushing old station entries
Deauthenticate all stations
Received 30 bytes management frame
  dump: b0 00 02 01 00 05 5d d9 8d ae 00 05 5d d9 57 59 00 05 5d d9 8d
ae a0 01 00 00 01 00 00 00
MGMT
mgmt::auth
authentication: STA=00:05:5d:d9:57:59 auth_alg=0 auth_transaction=1
status_code=0
  New STA
Station 00:05:5d:d9:57:59 authentication OK (open system)
Received 30 bytes management frame
  dump: b2 00 3a 01 00 05 5d d9 57 59 00 05 5d d9 8d ae 00 05 5d d9 8d
ae 20 44 00 00 02 00 00 00
MGMT (TX callback) ACK
mgmt::auth cb
Station 00:05:5d:d9:57:59 authenticated
Received 40 bytes management frame
  dump: 00 00 02 01 00 05 5d d9 8d ae 00 05 5d d9 57 59 00 05 5d d9 8d
ae b0 01 01 00 01 00 00 04 74 65 73 74 01 04 82 84 0b 16
MGMT
mgmt::assoc_req
association request: STA=00:05:5d:d9:57:59 capab_info=0x01
listen_interval=1
  new AID 1
Station 00:05:5d:d9:57:59 association OK (aid 1)
Received 36 bytes management frame
  dump: 12 00 3a 01 00 05 5d d9 57 59 00 05 5d d9 8d ae 00 05 5d d9 8d
ae 30 44 01 00 00 00 01 c0 01 04 82 84 0b 16
MGMT (TX callback) ACK
mgmt::assoc_resp cb
Station 00:05:5d:d9:57:59 associated (aid 1)
IEEE 802.1X: Start authentication for new station 00:05:5d:d9:57:59
IEEE 802.1X: 00:05:5d:d9:57:59 AUTH_PAE entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 AUTH_KEY_TX entering state
NO_KEY_TRANSMIT
IEEE 802.1X: 00:05:5d:d9:57:59 AUTH_PAE entering state DISCONNECTED
IEEE 802.1X: Unauthorizing station 00:05:5d:d9:57:59
IEEE 802.1X: Sending canned EAP packet FAILURE to 00:05:5d:d9:57:59
(identifier 0)
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state IDLE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 AUTH_PAE entering state CONNECTING
IEEE 802.1X: Sending EAP Request-Identity to 00:05:5d:d9:57:59
(identifier 1)
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
```

```
Received 40 bytes management frame
  dump: 0a 02 3a 01 00 05 5d d9 57 59 00 05 5d d9 8d ae 00 05 5d d9 8d
ae 40 44 aa aa 03 00 00 00 88 8e 01 00 00 04 04 00 00 04
DATA (TX callback) ACK
Received 46 bytes management frame
  dump: 0a 02 3a 01 00 05 5d d9 57 59 00 05 5d d9 8d ae 00 05 5d d9 8d
ae 50 44 aa aa 03 00 00 00 88 8e 01 00 00 0a 01 01 00 0a 01 68 65 6c 6c
6f
DATA (TX callback) ACK
Received 37 bytes management frame
  dump: 08 01 02 01 00 05 5d d9 8d ae 00 05 5d d9 57 59 00 05 5d d9 8d
ae c0 01 aa aa 03 00 00 00 88 8e 01 01 00 00 00
DATA
IEEE 802.1X: 5 bytes from 00:05:5d:d9:57:59
   IEEE 802.1X: version=1 type=1 length=0
   ignoring 1 extra octets after IEEE 802.1X packet
   EAPOL-Start
IEEE 802.1X: 00:05:5d:d9:57:59 AUTH_PAE entering state CONNECTING
IEEE 802.1X: Sending EAP Request-Identity to 00:05:5d:d9:57:59
(identifier 1)
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 46 bytes management frame
  dump: 0a 02 3a 01 00 05 5d d9 57 59 00 05 5d d9 8d ae 00 05 5d d9 8d
ae 60 44 aa aa 03 00 00 00 88 8e 01 00 00 0a 01 01 00 0a 01 68 65 6c 6c
6f
DATA (TX callback) ACK
Received 52 bytes management frame
  dump: 08 01 02 01 00 05 5d d9 8d ae 00 05 5d d9 57 59 00 05 5d d9 8d
ae d0 01 aa aa 03 00 00 00 88 8e 01 00 00 10 02 01 00 10 01 6e 65 77 78
70 63 6c 69 65 6e 74
DATA
IEEE 802.1X: 20 bytes from 00:05:5d:d9:57:59
   IEEE 802.1X: version=1 type=0 length=16
   EAP: code=2 identifier=1 length=16 (response)
   EAP Response-Identity
IEEE 802.1X: 00:05:5d:d9:57:59 AUTH_PAE entering state AUTHENTICATING
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
Sending RADIUS message to authentication server
RADIUS message: code=1 (Access-Request) identifier=0 length=160
   Attribute 1 (User-Name) length=13
      Value: 'newxpclient'
   Attribute 4 (NAS-IP-Address) length=6
      Value: 131.120.8.145
   Attribute 5 (NAS-Port) length=6
      Value: 1
   Attribute 30 (Called-Station-Id) length=24
      Value: '00-05-5D-D9-8D-AE:test'
   Attribute 31 (Calling-Station-Id) length=19
      Value: '00-05-5D-D9-57-59'
   Attribute 12 (Framed-MTU) length=6
      Value: 2304
   Attribute 61 (NAS-Port-Type) length=6
      Value: 19
   Attribute 77 (Connect-Info) length=24
      Value: 'CONNECT 11Mbps 802.11b'
```

```
   Attribute 79 (EAP-Message) length=18
   Attribute 80 (Message-Authenticator) length=18
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 52 bytes management frame
  dump: 08 01 02 01 00 05 5d d9 8d ae 00 05 5d d9 57 59 00 05 5d d9 8d
ae e0 01 aa aa 03 00 00 00 88 8e 01 00 00 10 02 01 00 10 01 6e 65 77 78
70 63 6c 69 65 6e 74
DATA
IEEE 802.1X: 20 bytes from 00:05:5d:d9:57:59
   IEEE 802.1X: version=1 type=0 length=16
   EAP: code=2 identifier=1 length=16 (response)
   EAP Response-Identity
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 64 bytes from authentication server
Received RADIUS message
RADIUS message: code=11 (Access-Challenge) identifier=0 length=64
   Attribute 79 (EAP-Message) length=8
   Attribute 80 (Message-Authenticator) length=18
   Attribute 24 (State) length=18
RADIUS packet matching with station 00:05:5d:d9:57:59
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:05:5d:d9:57:59 (identifier 2)
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 42 bytes management frame
  dump: 0a 02 3a 01 00 05 5d d9 57 59 00 05 5d d9 8d ae 00 05 5d d9 8d
ae 90 44 aa aa 03 00 00 00 88 8e 01 00 00 06 01 02 00 06 0d 20
DATA (TX callback) ACK
Received 148 bytes management frame
  dump: 08 01 02 01 00 05 5d d9 8d ae 00 05 5d d9 57 59 00 05 5d d9 8d
ae f0 01 aa aa 03 00 00 00 88 8e 01 00 00 70 02 02 00 70 0d 80 00 00 00
66 16 03 01 00 61 01 00 00 5d 03 01 40 06 d1 58 59 2c af d8 f8 1e 81 19
ca 6e c5 66 34 d6 a6 28 85 47 61 eb 69 e8 c9 3c 8f a4 a0 00 20 81 90 a4
11 52 ad 3b 0b 8f f1 cd 8a 98 ce 08 51 41 c8 f5 75 34 35 54 84 9b 7a 08
f5 73 5d d0 82 00 16 00 04 00 05 00 0a 00 09 00 64 00 62 00 03 00 06 00
13 00 12 00 63 01 00
DATA
IEEE 802.1X: 116 bytes from 00:05:5d:d9:57:59
   IEEE 802.1X: version=1 type=0 length=112
   EAP: code=2 identifier=2 length=112 (response)
   EAP Response-TLS
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
Sending RADIUS message to authentication server
RADIUS message: code=1 (Access-Request) identifier=1 length=274
   Attribute 1 (User-Name) length=13
      Value: 'newxpclient'
   Attribute 4 (NAS-IP-Address) length=6
      Value: 131.120.8.145
   Attribute 5 (NAS-Port) length=6
      Value: 1
   Attribute 30 (Called-Station-Id) length=24
      Value: '00-05-5D-D9-8D-AE:test'
   Attribute 31 (Calling-Station-Id) length=19
      Value: '00-05-5D-D9-57-59'
   Attribute 12 (Framed-MTU) length=6
```

```
      Value: 2304
   Attribute 61 (NAS-Port-Type) length=6
      Value: 19
   Attribute 77 (Connect-Info) length=24
      Value: 'CONNECT 11Mbps 802.11b'
   Attribute 79 (EAP-Message) length=114
   Attribute 24 (State) length=18
   Attribute 80 (Message-Authenticator) length=18
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 1100 bytes from authentication server
Received RADIUS message
RADIUS message: code=11 (Access-Challenge) identifier=1 length=1100
   Attribute 79 (EAP-Message) length=255
   Attribute 79 (EAP-Message) length=255
   Attribute 79 (EAP-Message) length=255
   Attribute 79 (EAP-Message) length=255
   Attribute 79 (EAP-Message) length=24
   Attribute 80 (Message-Authenticator) length=18
   Attribute 24 (State) length=18
RADIUS packet matching with station 00:05:5d:d9:57:59
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:05:5d:d9:57:59 (identifier 3)
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 1070 bytes management frame
  dump: 0a 02 3a 01 00 05 5d d9 57 59 00 05 5d d9 8d ae 00 05 5d d9 8d
ae a0 44 aa aa 03 00 00 00 88 8e 01 00 04 0a 01 03 04 0a 0d c0 00 00 06
b9 16 03 01 00 4a 02 00 00 46 03 01 40 06 d2 6c 7e f5 d8 69 c3 00 ae 17
26 a7 d7 0d 99 41 70 d4 6d 73 40 df b3 7b 42 95 c9 f5 29 1d 20 07 81 d9
93 df a6 c5 e5 ca d6 d1 c5 7c 5f ed 8b 54 c0 04 e5 30 71 f2 80 7f 8f 9c
e8 41 f1 40 fa 00 04 00 16 03 01 05 d9 0b 00 05 d5 00 05 d2 00 02 92 30
82 02 8e 30 82 01 f7 a0 03 02 01 02 02 01 01 30 0d 06 09 2a 86 48 86 f7
0d 01 01 04 05 00 30 76 31 0b 30 09 06 03 55 04 06 13 02 55 53 31 13 30
11 06 03 55 04 08 13 0a 43 61 6c 69 66 6f 72 6e 69 61 31 11 30 0f 06 03
55 04 07 13 08 4d 6f 6e 74 65 72 65 79 31 0d 30 0b 06 03 55 04 0a 13 04
4e 50 47 53 31 0d 30 0b 06 03 55 04 0b 13 04 53 41 41 4d 31 21 30 1f 06
09 2a 86 48 86 f7 0d 01 09 01 16 12 6f 6f 7a 61 6e 40 6e 70 73 2e 6e 61
76 79 2e 6d 69 6c 30 1e 17 0d 30 34 30 31 31 35 30 30 33 39 31 32 5a 17
0d 30 35 30 31 31 34 30 30 33 39 31 32 5a 30 81 8a 31 0b 30 09 06 03 55
04 06 13 02 55 53 31 13 30 11 06 03 55 04 08 13 0a 43 61 6c 69 66 6f 72
6e 69 61 31 11 30 0f 06 03 55 04 07 13 08 4d 6f 6e 74 65 72 65 79 31 0d
30 0b 06 03 55 04 0a 13 04 4e 50 47 53 31 0d 30 0b 06 03 55 04 0b 13 04
53 41 41 4d 31 12 30 10 06 03 55 04 03 13 09 6e 65 77 72 61 64 69 75 73
31 21 30 1f 06 09 2a 86 48 86 f7 0d 01 09 01 16 12 6f 6f 7a 61 6e 40 6e
70 73 2e 6e 61 76 79 2e 6d 69 6c 30 81 9f 30 0d 06 09 2a 86 48 86 f7 0d
01 01 01 05 00 03 81 8d 00 30 81 89 02 81 81 00 c7 c2 5b ce 6b b1 44 b7
1d e6 f3 8a 99 76 ae 25 ec 70 68 3e ef 15 03 63 14 0b 3c 95 f1 fc 4e d8
6e b7 ed 33 85 93 f8 4b ed c5 b8 91 e4 ff 1f eb 93 85 e1 4e ba 1a f3 c6
b5 79 fe b1 19 c0 89 63 73 07 13 3f f7 3b 97 f5 3f 72 fd 6a f6 e2 3c 28
56 c4 45 56 e7 b0 fb 6d 4f 60 94 94 10 96 af 1c 84 f4 91 e6 0f d5 61 17
a8 b7 05 45 b9 17 dd 14 8c 84 d5 38 9a 63 e1 66 4f 87 b1 19 17 98 cb 75
02 03 01 00 01 a3 17 30 15 30 13 06 03 55 1d 25 04 0c 30 0a 06 08 2b 06
01 05 05 07 03 01 30 0d 06 09 2a 86 48 86 f7 0d 01 01 04 05 00 03 81 81
00 07 c2 b6 90 91 fd 0a 1e 8f c1 98 41 a9 9d 9e d2 36 37 24 97 4f f6 91
eb 95 44 45 37 95 72 96 9a 90 71 0c 9e cc 62 36 28 0d 07 2d 8d e0 30 81
20 af 7d e2 33 2e 46 6d f5 6f 72 28 90 c0 68 eb 4b 51 72 3a 52 e6 6b 23
```

```
80 94 6f 86 81 2e 3b 71 d3 15 ab 90 5c ad 06 51 0e 6b a1 fa 6e d9 0c e0
45 f3 9b ab 76 7e ab 63 94 73 bb a6 e8 d9 e2 fb e2 cb e1 3d 57 56 31 fa
a6 de e2 61 a4 48 7f e9 ef 00 03 3a 30 82 03 36 30 82 02 9f a0 03 02 01
02 02 01 00 30 0d 06 09 2a 86 48 86 f7 0d 01 01 04 05 00 30 76 31 0b 30
09 06 03 55 04 06 13 02 55 53 31 13 30 11 06 03 55 04 08 13 0a 43 61 6c
69 66 6f 72 6e 69 61 31 11 30 0f 06 03 55 04 07 13 08 4d 6f 6e 74 65 72
65 79 31 0d 30 0b 06 03 55 04 0a 13 04 4e 50 47 53 31 0d 30 0b 06 03 55
04 0b 13 04 53 41 41 4d 31 21 30 1f 06 09 2a 86 48 86 f7 0d 01 09 01 16
12 6f 6f 7a 61 6e 40 6e 70 73 2e 6e 61 76 79 2e 6d 69 6c 30 1e 17 0d 30
34 30 31 31 35 30 30 33 33 33 37 5a 17 0d 30 34 30 32 31 34 30 30 33 33
33 37 5a 30 76 31 0b 30 09 06 03 55 04 06 13 02 55 53 31 13 30 11 06 03
55 04 08 13 0a 43 61 6c 69 66 6f 72 6e 69 61 31 11 30 0f 06 03 55 04 07
13 08 4d 6f 6e 74 65 72 65 79 31 0d 30 0b 06 03 55 04 0a 13 04 4e 50 47
53 31 0d 30 0b 06 03 55 04 0b 13 04 53 41 41 4d 31
DATA (TX callback) ACK
Received 42 bytes management frame
  dump: 08 01 02 01 00 05 5d d9 8d ae 00 05 5d d9 57 59 00 05 5d d9 8d
ae 00 02 aa aa 03 00 00 00 88 8e 01 00 00 06 02 03 00 06 0d 00
DATA
IEEE 802.1X: 10 bytes from 00:05:5d:d9:57:59
   IEEE 802.1X: version=1 type=0 length=6
   EAP: code=2 identifier=3 length=6 (response)
   EAP Response-TLS
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
Sending RADIUS message to authentication server
RADIUS message: code=1 (Access-Request) identifier=2 length=168
   Attribute 1 (User-Name) length=13
      Value: 'newxpclient'
   Attribute 4 (NAS-IP-Address) length=6
      Value: 131.120.8.145
   Attribute 5 (NAS-Port) length=6
      Value: 1
   Attribute 30 (Called-Station-Id) length=24
      Value: '00-05-5D-D9-8D-AE:test'
   Attribute 31 (Calling-Station-Id) length=19
      Value: '00-05-5D-D9-57-59'
   Attribute 12 (Framed-MTU) length=6
      Value: 2304
   Attribute 61 (NAS-Port-Type) length=6
      Value: 19
   Attribute 77 (Connect-Info) length=24
      Value: 'CONNECT 11Mbps 802.11b'
   Attribute 79 (EAP-Message) length=8
   Attribute 24 (State) length=18
   Attribute 80 (Message-Authenticator) length=18
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 769 bytes from authentication server
-------
-------
-------
RADIUS packet matching with station 00:05:5d:d9:57:59
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:05:5d:d9:57:59 (identifier 15)
Received 743 bytes management frame
```

```
  dump: 0a 02 02 01 00 05 5d d9 57 59 00 05 5d d9 8d ae 00 05 5d d9 8d
ae 60 4c aa aa 03 00 00 00 88 8e 01 00 02 c3 01 0f 02 c3 0d 80 00 00 06
b9 21 30 1f 06 09 2a 86 48 86 f7 0d 01 09 01 16 12 6f 6f 7a 61 6e 40 6e
70 73 2e 6e 61 76 79 2e 6d 69 6c 30 81 9f 30 0d 06 09 2a 86 48 86 f7 0d
01 01 01 05 00 03 81 8d 00 30 81 89 02 81 81 00 cc fe 01 16 21 92 44 8b
98 4d 48 59 a5 9d ae 3d 8b 5c eb d1 3f df 0c 93 c9 70 75 a0 a6 d4 b8 d8
ff e7 83 c2 96 5e 14 09 47 aa da 91 f5 98 97 12 eb 37 42 81 6b 9f 6b 41
ce 41 9d f7 89 50 05 67 64 a4 bd d0 44 a1 06 d2 71 fc 31 01 e2 8f b4 06
5f b1 56 07 a3 c7 fd de 46 c6 a7 8d e6 65 db 4a f1 64 2b 48 b1 5e 51 3b
d9 a0 33 1b 71 db 7a 9f 3f ea ae fa 4c 65 d0 6c da 7e 44 ee a9 8e 4b 13
02 03 01 00 01 a3 81 d3 30 81 d0 30 1d 06 03 55 1d 0e 04 16 04 14 5f 18
3d 02 8f ea ae 3c 3a a6 a5 53 82 29 73 24 68 86 2b 6c 30 81 a0 06 03 55
1d 23 04 81 98 30 81 95 80 14 5f 18 3d 02 8f ea ae 3c 3a a6 a5 53 82 29
73 24 68 86 2b 6c a1 7a a4 78 30 76 31 0b 30 09 06 03 55 04 06 13 02 55
53 31 13 30 11 06 03 55 04 08 13 0a 43 61 6c 69 66 6f 72 6e 69 61 31 11
30 0f 06 03 55 04 07 13 08 4d 6f 6e 74 65 72 65 79 31 0d 30 0b 06 03 55
04 0a 13 04 4e 50 47 53 31 0d 30 0b 06 03 55 04 0b 13 04 53 41 41 4d 31
21 30 1f 06 09 2a 86 48 86 f7 0d 01 09 01 16 12 6f 6f 7a 61 6e 40 6e 70
73 2e 6e 61 76 79 2e 6d 69 6c 82 01 00 30 0c 06 03 55 1d 13 04 05 30 03
01 01 ff 30 0d 06 09 2a 86 48 86 f7 0d 01 01 04 05 00 03 81 81 00 52 89
a7 07 95 4e 9e 1c 2f be 3c c5 79 5b 66 0a 06 4b ab 0f 54 28 10 c3 2b 28
96 f0 53 66 06 bc 49 45 74 b2 e5 eb 31 cc b2 e5 bb 8e 74 60 be 48 c4 03
b6 2f dc c3 d5 79 6b 92 1d ef 8b 8e 20 26 7d 15 02 1b 96 a0 f6 4a 3e 46
3b 44 5e 17 dd 3e e4 dc ce e7 98 57 b3 7f 28 5a 9c ab 2f 68 e7 0e 80 98
d0 4e 30 1f 2c 30 bb aa 1f 50 fe 90 af 6d 7a 05 f0 23 a5 e1 f9 35 bb dc
57 32 5a a8 e1 b6 16 03 01 00 87 0d 00 00 7f 02 01 02 00 7a 00 78 30 76
31 0b 30 09 06 03 55 04 06 13 02 55 53 31 13 30 11 06 03 55 04 08 13 0a
43 61 6c 69 66 6f 72 6e 69 61 31 11 30 0f 06 03 55 04 07 13 08 4d 6f 6e
74 65 72 65 79 31 0d 30 0b 06 03 55 04 0a 13 04 4e 50 47 53 31 0d 30 0b
06 03 55 04 0b 13 04 53 41 41 4d 31 21 30 1f 06 09 2a 86 48 86 f7 0d 01
09 01 16 12 6f 6f 7a 61 6e 40 6e 70 73 2e 6e 61 76 79 2e 6d 69 6c 0e 00
00 00
DATA (TX callback) ACK
Received 1032 bytes management frame
  dump: 08 01 02 01 00 05 5d d9 8d ae 00 05 5d d9 57 59 00 05 5d d9 8d
ae e0 04 aa aa 03 00 00 00 88 8e 01 00 03 e4 02 0f 03 e4 0d 80 00 00 03
da 16 03 01 03 aa 0b 00 02 9a 00 02 97 00 02 94 30 82 02 90 30 82 01 f9
a0 03 02 01 02 02 01 02 30 0d 06 09 2a 86 48 86 f7 0d 01 01 04 05 00 30
76 31 0b 30 09 06 03 55 04 06 13 02 55 53 31 13 30 11 06 03 55 04 08 13
0a 43 61 6c 69 66 6f 72 6e 69 61 31 11 30 0f 06 03 55 04 07 13 08 4d 6f
6e 74 65 72 65 79 31 0d 30 0b 06 03 55 04 0a 13 04 4e 50 47 53 31 0d 30
0b 06 03 55 04 0b 13 04 53 41 41 4d 31 21 30 1f 06 09 2a 86 48 86 f7 0d
01 09 01 16 12 6f 6f 7a 61 6e 40 6e 70 73 2e 6e 61 76 79 2e 6d 69 6c 30
1e 17 0d 30 34 30 31 31 35 30 30 34 30 32 37 5a 17 0d 30 35 30 31 31 34
30 30 34 30 32 37 5a 30 81 8c 31 0b 30 09 06 03 55 04 06 13 02 55 53 31
13 30 11 06 03 55 04 08 13 0a 43 61 6c 69 66 6f 72 6e 69 61 31 11 30 0f
06 03 55 04 07 13 08 4d 6f 6e 74 65 72 65 79 31 0d 30 0b 06 03 55 04 0a
13 04 4e 50 47 53 31 0d 30 0b 06 03 55 04 0b 13 04 53 41 41 4d 31 14 30
12 06 03 55 04 03 13 0b 6e 65 77 78 70 63 6c 69 65 6e 74 31 21 30 1f 06
09 2a 86 48 86 f7 0d 01 09 01 16 12 6f 6f 7a 61 6e 40 6e 70 73 2e 6e 61
76 79 2e 6d 69 6c 30 81 9f 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00
03 81 8d 00 30 81 89 02 81 81 00 b6 2b a4 23 42 e1 59 dd 8d cc ba d2 28
50 b3 eb ed 4e c9 1f 9d d3 83 56 34 ae bd c9 bf 3d df 49 32 7c 0a ca 16
95 20 06 dd 77 13 0c b5 c2 e8 be dd d8 9c 94 e6 fb 6d 96 17 01 0a 02 59
d2 20 3f 79 5d ea 16 99 25 69 46 47 7e 15 49 54 13 c4 38 4d 83 ff b6 1b
bd 13 c6 e2 93 12 17 2a 5b 9a 3a 48 53 59 76 98 04 75 30 06 93 65 75 86
00 01 fe 90 09 17 74 40 7a 71 fc 6f 97 67 a6 ff 60 66 bf 02 03 01 00 01
```

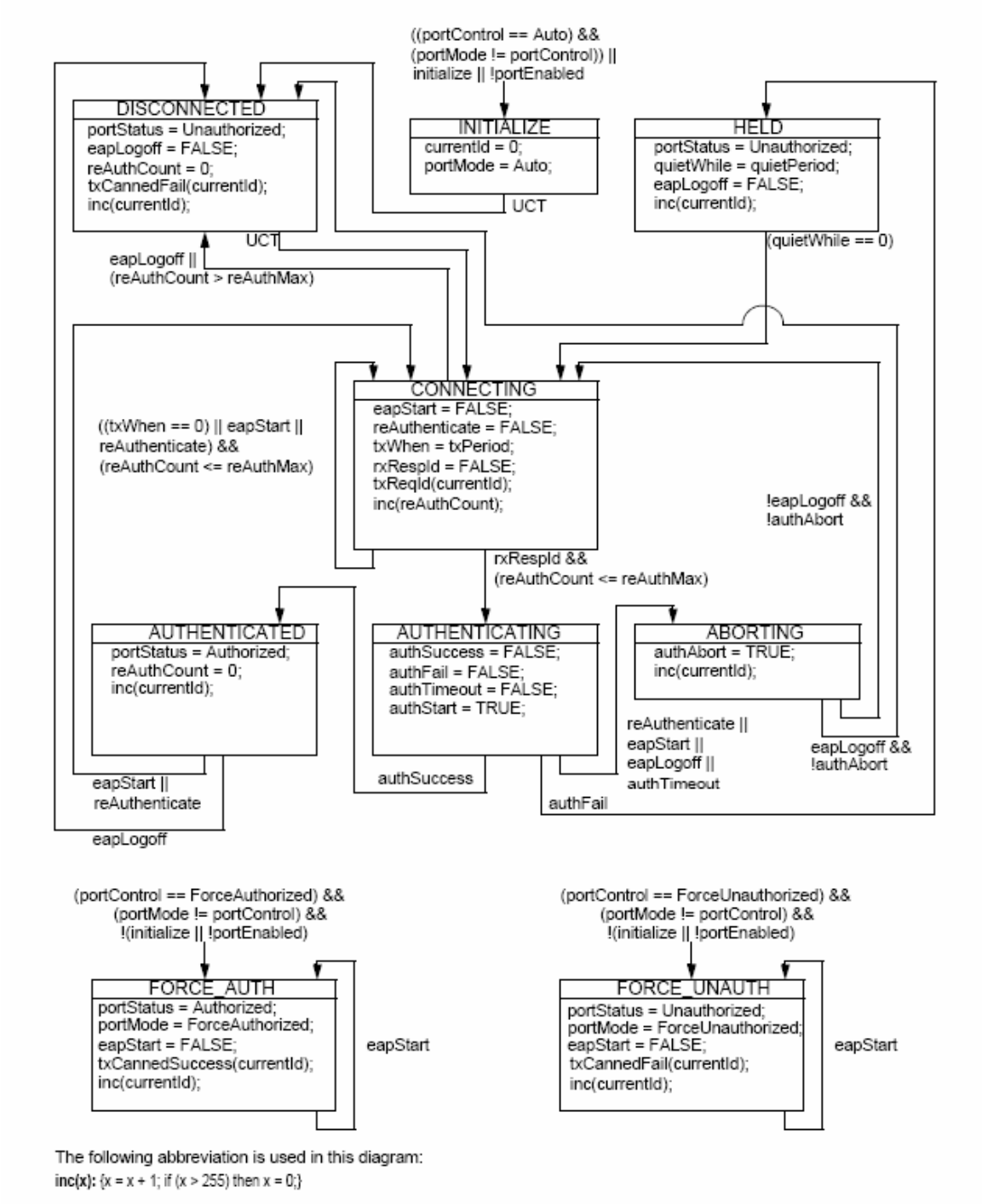
```
a3 17 30 15 30 13 06 03 55 1d 25 04 0c 30 0a 06 08 2b 06 01 05 05 07 03
02 30 0d 06 09 2a 86 48 86 f7 0d 01 01 04 05 00 03 81 81 00 c1 89 97 3d
d1 f6 63 2d ef 5a 12 3a 39 99 0e 8e 41 f3 ea ee be a6 45 e1 f4 2b 58 22
ea ce 5d 08 19 dd a9 3d 55 6c 87 44 03 00 d8 ca f1 02 a7 88 0c 33 20 6f
d7 be 6c 2c 14 32 c6 c7 b7 c9 3a 2f fb 4e d1 97 ed 00 ff 9f 78 c2 75 b8
44 42 72 06 7c 17 53 46 03 03 a0 e1 f4 58 f7 83 bf cd c2 17 d9 d9 8e 78
70 20 a5 6a 40 b9 94 34 fe 3a f4 d8 fc 3e 9f 3e f7 4d c1 09 7f 65 48 75
3f d7 4d 5c 10 00 00 82 00 80 06 56 1e 5e 9d 9e 9c 8b 9f 60 3a 3d 6b 1f
13 82 82 df e6 11 2e 12 ef c1 0c 0a c5 dd 45 22 78 60 2f ca 4b 97 51 3a
89 6c 09 8f a0 d5 b6 7b 7d bd 7e 2e d6 c9 21 53 b6 1a 5f 92 ff ab e0 a3
64 1c 85 06 8c 5d 03 3d 78 37 2a 09 2a 41 6a f4 0f c4 de a9 d1 e5 03 1e
f9 79 aa ab 56 61 89 fe 82 a0 1b 50 92 88 22 00 fc 45 72 18 55 84 d6 ed
57 8f 6e 39 f8 42 5b cd 59 5d 55 a4 68 fd 0d da 1b 2f 0f 00 00 82 00 80
23 44 69 5b 98 67 df 6e b6 4e 52 41 a0 51 4e 2f 63 b1 24 0b d3 76 e4 53
d6 54 9f 30 52 4d 0a d9 ad e3 7d 41 cd da b5 a2 90 90 7e f7 dd 91 56 64
97 b6 26 7a 33 45 59 c6 de 78 86 6a db 9d 18 1b bc e8 67 4f 37 1e ac de
c9 1f 5c 30 8b be 4c d2 94 55 d1 c2 5a 01 40 4c 91 b1 c1 27 1d 5e be 39
40 b8 47 e9 4d 21 8f 56 3e f4 ed 37 a6 7d 9d e8 c0 de 36 59 2d 37 75 1b
1f a8 71 d9 6f 93 e0 f6 14 03 01 00 01 01 16 03 01 00 20 88 a6 dd 59 65
b0 d8 6d b0 88 e9 ee ae 76 37 00 49 b9 c5 3a 0f b1 6c 85 c5 fa 48 29 3a
9e 3f 7d
DATA
IEEE 802.1X: 1000 bytes from 00:05:5d:d9:57:59
   IEEE 802.1X: version=1 type=0 length=996
   EAP: code=2 identifier=15 length=996 (response)
   EAP Response-TLS
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
Sending RADIUS message to authentication server
RADIUS message: code=1 (Access-Request) identifier=12 length=1164
   Attribute 1 (User-Name) length=13
      Value: 'newxpclient'
   Attribute 4 (NAS-IP-Address) length=6
      Value: 131.120.8.145
   Attribute 5 (NAS-Port) length=6
      Value: 1
   Attribute 30 (Called-Station-Id) length=24
      Value: '00-05-5D-D9-8D-AE:test'
   Attribute 31 (Calling-Station-Id) length=19
      Value: '00-05-5D-D9-57-59'
   Attribute 12 (Framed-MTU) length=6
      Value: 2304
   Attribute 61 (NAS-Port-Type) length=6
      Value: 19
   Attribute 77 (Connect-Info) length=24
      Value: 'CONNECT 11Mbps 802.11b'
   Attribute 79 (EAP-Message) length=255
   Attribute 79 (EAP-Message) length=255
   Attribute 79 (EAP-Message) length=255
   Attribute 79 (EAP-Message) length=239
   Attribute 24 (State) length=18
   Attribute 80 (Message-Authenticator) length=18
Received 111 bytes from authentication server
Received RADIUS message
RADIUS message: code=11 (Access-Challenge) identifier=12 length=111
   Attribute 79 (EAP-Message) length=55
   Attribute 80 (Message-Authenticator) length=18
```

```
   Attribute 24 (State) length=18
RADIUS packet matching with station 00:05:5d:d9:57:59
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:05:5d:d9:57:59 (identifier 16)
Received 89 bytes management frame
  dump: 0a 02 02 01 00 05 5d d9 57 59 00 05 5d d9 8d ae 00 05 5d d9 8d
ae c0 4c aa aa 03 00 00 00 88 8e 01 00 00 35 01 10 00 35 0d 80 00 00 00
2b 14 03 01 00 01 01 16 03 01 00 20 e4 a4 01 c9 fe 54 a2 12 4b 7c 5b 32
f5 e5 b2 7e 32 bc 67 46 4b 52 bc dc b9 03 a7 8d 1f d1 71 28
DATA (TX callback) ACK
Received 42 bytes management frame
  dump: 08 01 02 01 00 05 5d d9 8d ae 00 05 5d d9 57 59 00 05 5d d9 8d
ae f0 04 aa aa 03 00 00 00 88 8e 01 00 00 06 02 10 00 06 0d 00
DATA
IEEE 802.1X: 10 bytes from 00:05:5d:d9:57:59
   IEEE 802.1X: version=1 type=0 length=6
   EAP: code=2 identifier=16 length=6 (response)
   EAP Response-TLS
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
Sending RADIUS message to authentication server
RADIUS message: code=1 (Access-Request) identifier=13 length=168
   Attribute 1 (User-Name) length=13
      Value: 'newxpclient'
   Attribute 4 (NAS-IP-Address) length=6
      Value: 131.120.8.145
   Attribute 5 (NAS-Port) length=6
      Value: 1
   Attribute 30 (Called-Station-Id) length=24
      Value: '00-05-5D-D9-8D-AE:test'
   Attribute 31 (Calling-Station-Id) length=19
      Value: '00-05-5D-D9-57-59'
   Attribute 12 (Framed-MTU) length=6
      Value: 2304
   Attribute 61 (NAS-Port-Type) length=6
      Value: 19
   Attribute 77 (Connect-Info) length=24
      Value: 'CONNECT 11Mbps 802.11b'
   Attribute 79 (EAP-Message) length=8
   Attribute 24 (State) length=18
   Attribute 80 (Message-Authenticator) length=18
Received 173 bytes from authentication server
Received RADIUS message
RADIUS message: code=2 (Access-Accept) identifier=13 length=173
   Attribute 26 (Vendor-Specific) length=58
   Attribute 26 (Vendor-Specific) length=58
   Attribute 79 (EAP-Message) length=6
   Attribute 80 (Message-Authenticator) length=18
   Attribute 1 (User-Name) length=13
      Value: 'newxpclient'
RADIUS packet matching with station 00:05:5d:d9:57:59
MS-MPPE-Send-Key (len=32): 1f 44 ce 96 1e cf 53 3c 72 8e 73 1d ff 14 4b
91 8e d1 a4 20 fd 83 18 5e 4e cc 6b 3c 68 6a 08 b9
MS-MPPE-Recv-Key (len=32): e9 54 5b 18 09 75 cd fb 5d 0f 98 21 89 e0 3b
43 60 2f 6c 47 5e 4e e6 6d 7d 24 78 3c 05 6f 31 4c
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state SUCCESS
IEEE 802.1X: Sending EAP Packet to 00:05:5d:d9:57:59 (identifier 16)
```

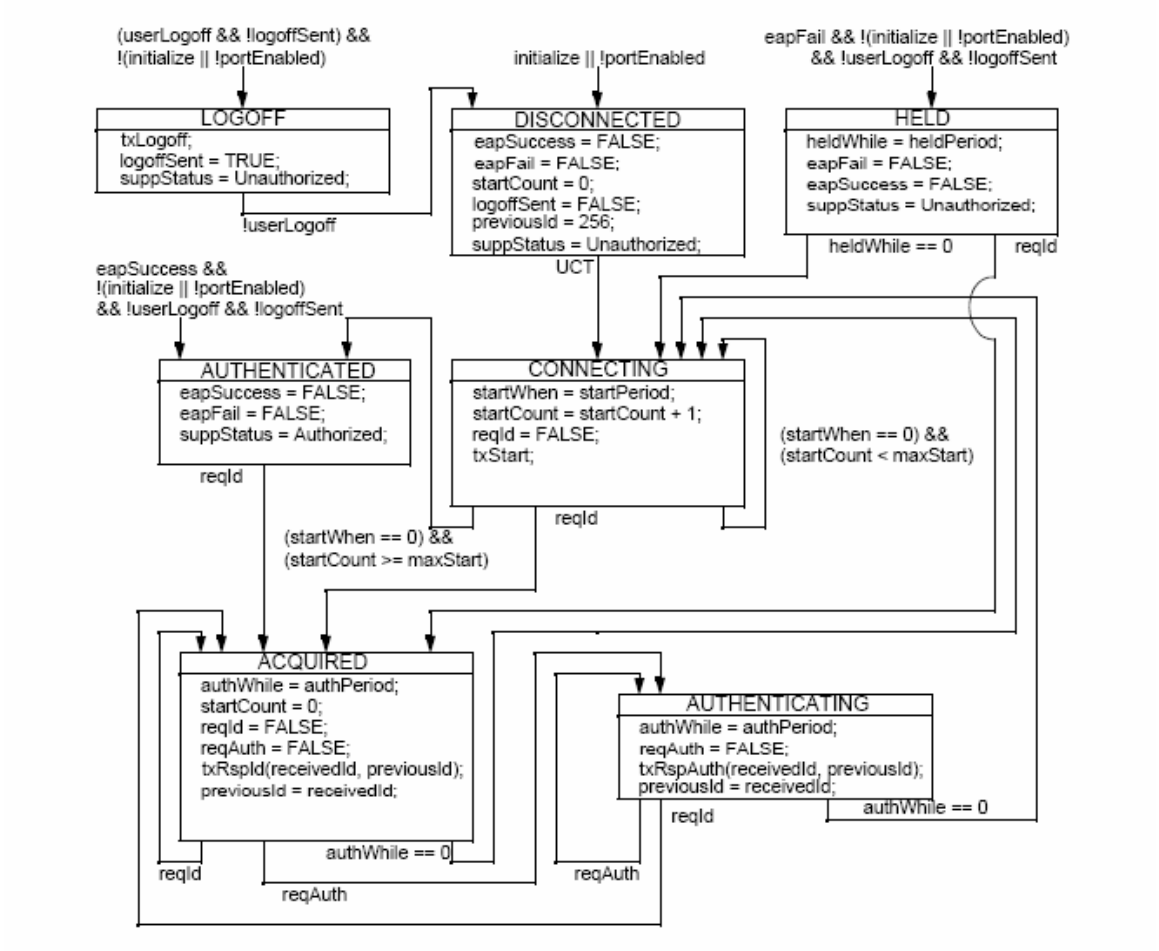
```
IEEE 802.1X: 00:05:5d:d9:57:59 AUTH_KEY_TX entering state KEY_TRANSMIT
IEEE 802.1X: Sending EAPOL-Key(s) to 00:05:5d:d9:57:59 (identifier 16)
IEEE 802.1X: Sending EAPOL-Key to 00:05:5d:d9:57:59 (broadcast index=1)
Individual WEP key - hexdump(len=5): bc b3 d0 ce 1d
IEEE 802.1X: Sending EAPOL-Key to 00:05:5d:d9:57:59 (unicast index=0)
IEEE 802.1X: 00:05:5d:d9:57:59 AUTH_PAE entering state AUTHENTICATED
IEEE 802.1X: Authorizing station 00:05:5d:d9:57:59
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state IDLE
Received 40 bytes management frame
  dump: 0a 02 02 01 00 05 5d d9 57 59 00 05 5d d9 8d ae 00 05 5d d9 8d
ae d0 4c aa aa 03 00 00 00 88 8e 01 00 00 04 03 10 00 04
DATA (TX callback) ACK
Received 85 bytes management frame
  dump: 0a 02 02 01 00 05 5d d9 57 59 00 05 5d d9 8d ae 00 05 5d d9 8d
ae e0 4c aa aa 03 00 00 00 88 8e 01 03 00 31 01 00 05 c3 b1 5b 4f 0a d4
90 d2 91 9f 41 bf 1c 8d 9a 89 2a 58 94 49 e0 d9 c5 c1 01 10 87 ca 61 01
32 8f 1f 86 0c e8 3a ad e6 c3 8f 6b c3 e5 3c 43
DATA (TX callback) ACK
Received 85 bytes management frame
  dump: 0a 02 02 01 00 05 5d d9 57 59 00 05 5d d9 8d ae 00 05 5d d9 8d
ae f0 4c aa aa 03 00 00 00 88 8e 01 03 00 31 01 00 05 c3 b1 5b 4f 0e 11
98 94 8d 17 4f 4f 99 c0 cb cd 9b 41 c1 76 72 bd c6 15 80 dd cc b2 db 32
7f 5c 65 72 90 18 32 b1 4b 57 1f 5e ed 7c 60 35
DATA (TX callback) ACK
IEEE 802.1X: 00:05:5d:d9:57:59 Port Timers TICK (timers: 29 0 3595 28)
IEEE 802.1X: 00:05:5d:d9:57:59 Port Timers TICK (timers: 28 0 3594 27)
```

APPENDIX F

A. AUTHENTICATOR STATE MACHINE



B. SUPPLICANT STATE MACHINE



LIST OF REFERENCES

1. A. Mishra, W. Arbaugh, "*An Initial Security Analysis of the IEEE 802.1X Standard"*, CS-TR-4328 UMIACS-TR-2002-10 Technical Report 6 February 2002.

2. Cisco Product Bulletin No.1327*"Cisco Comments on Recent WLAN Security Paper from University of Maryland"*, 6 October 2001.

3. Agere Systems, *"ORINOCO WLAN Security Response to 'An Initial Security Analysis of the IEEE 802.1X Standard', W.Arbaugh and A,Mishra, University of Maryland"* December 2001

4. H.Selcuk Ozturk *"Evaluation of Secure 802.1X Part-Based Network Access Authentication Over 802.11 Wireless Local Area Networks"*, Naval Postgraduate School, CA 93943, March 2003

5. Geier Jim, *Wireless LANs Implementing Interoperable Networks*, Macmillan Network Architecture & Development Series, 1999.

6. Behrouz A. Forozuan, *Local Area Networks*, McGraw Hill, 2003.

7. Institute of Electrical and Electronics Engineers, "*ANSI/IEEE Std 802.11 Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications"*, 20 August 1999.

8. Institute of Electrical and Electronics Engineers, "*ANSI/IEEE Std 802.11b/D8.0, DRAFT Supplement to STANDART for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Higher-Speed Physical Layer Extension in the 2.4 GHZ Band"*, September 2001.

9. Certified Wireless Network Administrator Official Study Guide, Planet3 2000.

10. Scott Fluhrer, Itsik Mantin, and Adi Shamir*, "Weakness in the Key Scheduling Algotrithm of RC4"*, SAC 2001.

11. Adam Stubblefield, John Ioannidis, and Aviel D. Rubin, "*Using Fluhrer, Mantin, and Shamir Attack to Break WEP"*, AT&T Labs Technical Report 2001.

12. Institute of Electrical and Electronics Engineers, *IEEE Standard for Local and Metropolitan Area Networks Port-Based Network Access Control*, IEEE Std 802.1X-2001.

13. W. Stallings, Wireless Communications and Networks, Prentice Hall 2002.

14. Jesse R. Walker, *"Unsafe at Any Key Size: An analysis of the WEP Encapsulation"*, doc: IEEE 802.11-00/362, October 2000.

15. Nikita Borisov, Ian Goldberg, and David Wagner, "*Intercepting Mobile Communications: The Insecurity of 802.11"*, ACM SIGMOBILE 7/01 Rome, Italy, 2001 ACM ISBN 1-58113-422-3/01/07.

16. B.Aboba and D.Simon, "*PPP EAP Authentication Protocol",* RFC 2716 Experimental, October 1999

17. C. Rigney, S. Willens, A. Rubens, W. Simpson, "*Remote Authentication Dial In User Service"*, RFC2138 June 2000.

18. Bernard Aboba, "*IEEE 802.1X Pre-Authentication"*, doc.: IEEE 802.11-02/389r0 June 2002.

19. T.Dierks, C. Allen *"The TLS Protocol Version 1.0"*, RFC-2246 January 1999.

20. Cisco Systems, Inc *"A Comprehensive Review of 802.11 Wireless LAN Security and the Cisco Wireless Security Suite"*, White Paper, January 2002

21. M. Sutton, *"Hacking the Invisible Network, Insecurities in 802.1X",* iDEFENSE Labs., 10 July 2002.

22. J. Walker, *"802.11i Overview part-I and part-II*", Microsoft Power point Presentation, Intel Corporation, Jesse Walker, March 2003

23. B. Aboda, D. Simon, *"PPP EAP TLS Authentication Protocol"*, RFC-2716, October 1999.

24. C. Rigney, S. Willens, A. Rubens, W. Simpson, *"Remote Authentication Dial In User Service (RADIUS)"* RFC-2865, June 2000.

25. C. Rigney, *"RADIUS Accounting"*, RFC-2866, June 2000

26. Cisco Systems, Inc. *"Extensible Authentication Protocol Transport Layer Security Deployment Guide for Wireless LAN Networks "*, White Paper, November 2002

27. D. Eaton, (1 November 2002), *"Diving into the 802.11i Spec: A Tutorial"* [online], Available from: http://www.commsdesign.com/design library/cd/hn /OEG20021126S0003, [Accessed 12 January 2004]

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Geoffrey Xie
   Naval Postgraduate School
   Monterey, CA

4. Professor John Gibson
   Naval Postgraduate School
   Monterey, CA

5. Deniz Kuvvetleri Komutanligi
   Personel Daire Baskanligi
   06410 Bakanliklar
   Ankara, TURKEY

6. Bogazici Universitesi
   Muhendislik Fakultesi, Bilgisayar Muhendisligi
   34342 Bebek
   Istanbul, TURKEY

7. Ortadogu Teknik Universitesi
   Muhendislik Fakultesi, Bilgisayar Muhendisligi
   06531
   Ankara, TURKEY

8. Hulusi ONDER
   Hatip mahallesi, No:43/A, 58700, Zara
   Sivas, TURKEY